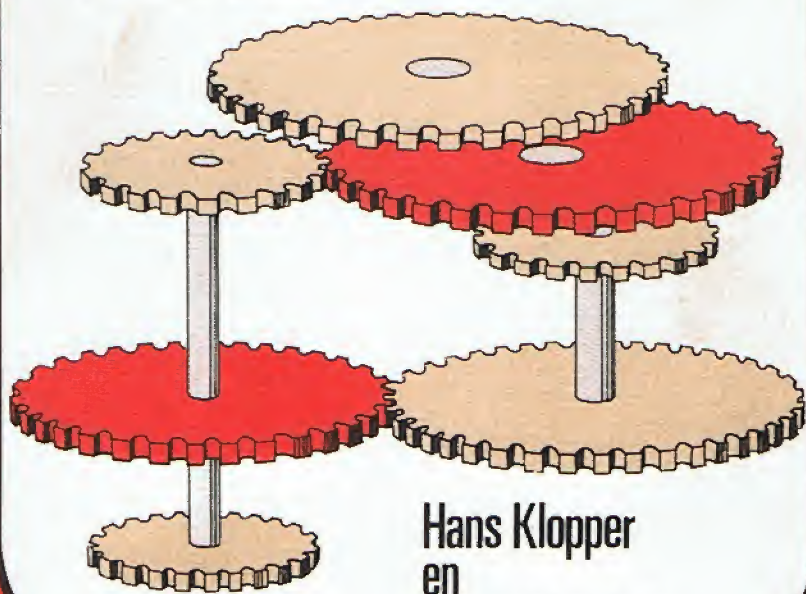
MSX

Machinetaal handboek



Hans Klopper
en
Marcel Le Belle



Machinetaal handboek

Hans Klopper en Marcel Le Belle

uitgeverij STARK-TEXEL

postbus 302 - 1794 ZG Oosterend tel. 02223 - 661

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Klopper, Hans

MSX Machinetaal handboek / Hans Klopper en Marcel Le Belle.
– Oosterend : Stark-Texel
ISBN 90-6398-735-8
SISO 365.3 UDC 681.3.06
Trefw.: MSX (computer) / programmeertalen.

1e druk 1986
ISBN 90 6398 735 8

INHOUD

Inleiding

Beste MSX gebruiker,

U heeft nu een boek in uw handen dat als doel heeft u een ruime inleiding te geven in machinetaal. Het moet nadat u dit boek goed heeft bestudeerd, mogelijk zijn om uw eigen machine code routines te ontwikkelen of eventueel een volledig programma in deze taal te schrijven. Er wordt vanuit gegaan, dat u BASIC vrij goed beheerst.

Waarom machine code?

Op deze vraag zijn verschillende antwoorden mogelijk.

- De snelheid (ongeveer 100 maal sneller dan een vergelijkbaar BASIC programma)

- De flexibiliteit (in machine code is vrijwel alles mogelijk)

- Groter beschikbaar vrij geheugen

Dit zijn de drie grootste voordelen van machine code t.o.v. BASIC. De snelheidsvoordelen zijn vooral zichtbaar in de zeer complexe spelen die zijn geschreven in deze taal. In BASIC zouden deze spelletjes niet om aan te zien zijn.

De flexibiliteit komt vooral tot uiting op

het moment dat u met BASIC dingen wilt doen die eigenlijk niet kunnen. Hoe worden bijvoorbeeld komplete beelden over het scherm verplaatst?
Dit doet men d.m.v. een techniek die men 'scrollen' noemt. Dit scrollen zullen wij in de loop van het boek verder uitdiepen.

In machinetaal heeft u in principe de beschikking over het volledige RAM geheugen van uw computer. Dit in tegenstelling tot BASIC, waarbij u maximaal over ongeveer 28K RAM beschikt.

Dit boek is verdeeld in twee belangrijke delen. Het eerste deel legt u uit hoe uw computer precies met zijn geheugen omspringt. Het is belangrijk voor een goed begrip van het tweede deel, dat over het feitelijke programmeren in machine code handelt. Mocht u niet voldoende informatie over de nodige RAM-adressen in dit boek kunnen vinden, dan verwijs ik u bij deze naar een zeer nuttig boek wat dit betreft, 'MSX verder uitgediept - Stark'. Hierin gaat de auteur o.a. zeer uitgebreid in op het Video RAM, de systeem variabelen, en het gebruik van ROM routines.

Onze dank gaat uit naar Electronics Nederland voor het beschikbaar stellen van twee SpectraVideo X' Press computers en het Wordstar tekstverwerkings programma. Hierdoor werd ons de tekstverwerking een stuk makkelijker gemaakt.

Nog een zeer belangrijke opmerking.

Als u iets niet in een keer helemaal be-

grijpt, lees datgene dàn nog eens overnieuw. Gebruik bij het invoeren van uw machine code programma's het liefst een zogenaamde 'assembler'. Dit is een programma dat ervoor zorgt dat u machinecode kunt invoeren in zogenaamde assemblertaal i.p.v. de standaard codes. Dit vergroot het gemak van het programmeren en de overzichtelijkheid van het programma. Goede assemblers voor MSX zijn:

Hisoft	-	Devpac	(cassette/disk)
Kuma	-	Zen	(cassette/disk)
Microsoft	-	M80	(disk)
Microsoft	-	Duad	(disk)

De verschillende talsystemen

Zoals u waarschijnlijk reeds weet, is uw MSX computer in staat om in verschillende talstelsels te werken. Het normale talstelsel waarmee we in het dagelijks leven omgaan, het zogenaamde 10-tallig stelsel kunt u in BASIC zeer goed gebruiken. Daarnaast zijn enkele andere talstelsels zoals het octale-, hexadecimale- of binaire talstelsel de bij MSX geboden alternatieven.

De computer werkt zelf met het binaire talstelsel. Als u in BASIC programeert worden door de in ROM aanwezige interpreter, al uw instrukties omgezet in binaire instrukties die de computer begrijpt. Daar dit stelsel voor ons mensen niet een van de meest eenvoudig te hanteren methodes is, geeft men vaak bij het programmeren in machinetaal de voorkeur aan het hexadecimale stelsel. Waarom? Dat zullen we zo zien. Het decimale stelsel zullen we bij het programmeren in machinetaal weinig gebruiken. Het octale stelsel tenslotte is het minst gebruikt bij MSX en we zullen het in onze programma'a ook niet toepassen.

We zullen nu elk van de talsystemen afzonderlijk bespreken.

Het decimale talstelsel

Het decimale systeem heeft als grondtal het getal 10. D.w.z., dat de getallen 0,1,2,3,4,5,6,7,8 en 9 de cijfers zijn waarmee alle mogelijke getallen gevormd kunnen worden.

Nemen we nu het getal 3750 als voorbeeld.

In dit getal zitten: 0 eenheden
5 tientallen
7 honderdtallen
3 duizendtallen

Duizend-tallen	Honderd-tallen	Tien-tallen	Een-heden
10^3	10^2	10^1	10^0

De meest rechtse positie voor het decimaalteken is de positie 0. Links daarvan is positie 1 etc.

Voor het getal 3750 vinden we nu.

$0 * 10^0 = 0$

$5 * 10^1 = 50$

$7 * 10^2 = 700$

$3 * 10^3 = 3000$

P.S.

- Elk getal dat tot de macht 0 wordt verheven, heeft de waarde 1.

Het binaire talstelsel

Zoals reeds vermeld, werkt de computer alleen binair. Het grondtal bij het binaire systeem is gelijk aan 2 (0 en 1). De nullen en eenen worden bits genoemd. Vier bits vormen een nibble. Acht bits vormen een byte.

Een binair getal is bijvoorbeeld: 1010 (10 decimaal)

Hoe is nu het verband tussen binair en decimaal?

```
1010 binair
!!!!- 0 * 2 tot de macht 0 = 0   bitnr. 0
!!!-- 1 * 2 tot de macht 1 = 2   bitnr. 1
!!--- 0 * 2 tot de macht 2 = 0   bitnr. 2
!---- 1 * 2 tot de macht 3 = 8   bitnr. 3
                 ------------- +
                            10 decimaal
```

Decimaal naar binair omzetting;

Hierbij moet u kijken welke macht van 2 als grootste voorkomt in het decimale getal, dat moet worden omgezet naar een binair getal. Dit getal trekt u dan van

het uitgangsgetal af. De uitkomst van die aftreksom bekijkt u dan weer opnieuw etc.

Voorbeeld: Het getal 329 decimaal

Het grootste getal is 256 = 2^8
329-256 =73

Het grootste getal is 64 = 2^6
73- 64 = 9

Het grootste getal is 8 = 2^3
9 - 8 = 1

In het getal 329 komen dus voor:
1 * 2 tot de macht 8 - bitnr. 8
0 * 2 tot de macht 7 - bitnr. 7
1 * 2 tot de macht 6 - bitnr. 6
0 * 2 tot de macht 5 - bitnr. 5
0 * 2 tot de macht 4 - bitnr. 4
1 * 2 tot de macht 3 - bitnr. 3
0 * 2 tot de macht 2 - bitnr. 2
0 * 2 tot de macht 1 - bitnr. 1
1 * 2 tot de macht 0 - bitnr. 0

Het binaire getal is nu: 101001001

In BASIC bestaan voor de verwerking van binaire getallen speciale voorzieningen. Een binair getal wordt genoteerd als:

```
&B101001001
```

Omzetten van decimaal naar binair:

```
PRINT BIN$(329) ==>> 101001001
```

Omzetten van binair naar decimaal:

PRINT &B101001001 ==>> 329

Binaire getallen gebruiken we in machinecode bijvoorbeeld als een bepaald register dat is opgebouwd uit 8 bits een bepaalde binaire waarde dient te krijgen om een functie te verwezenlijken.

Een byte;

```
-----------------------------------------
!bit !bit !bit !bit !bit !bit !bit !bit !
! 7  ! 6  ! 5  ! 4  ! 3  ! 2  ! 1  ! 0  !
-----------------------------------------
```

Voorbeeld: VDP-register 1 moet een zekere binaire waarde krijgen om de display aan of uit te schakelen. Het bit dat hiervoor aan of uit dient te worden geschakeld, is het zevende bit.

We zullen nu het voorbeeld in BASIC laten zien.

VDP(1)=&B10100010 ==>> DISPLAY UIT

VDP(1)=&B11100010 ==>> DISPLAY AAN

Tot zover het binaire talstelsel.

Het hexadecimale talstelsel

Bij het hexadecimale getallensysteem maken we gebruik van 16 als grondtal (0,1,2,3,4, 5,6,7,8,9,A,B,C,D,E,F).

In BASIC moeten hexadecimale getallen voorafgegaan worden met '&H'. Bij het gebruik van een assembler wordt er meestal een hoofdletter-H achter het hexadecimale getal geplaatst.

Een voorbeeld van een hexadecimaal getal:

FFH (255 decimaal)

FF is het getal, de H geeft aan dat het hier om een hexadecimaal getal gaat. U kunt hier al in een oogopslag zien dat het om een hexadecimaal getal gaat omdat er tweemaal de letter F in voorkomt. Ziet u echter ook het verschil tussen 20H en 20 als er geen H achter het hexadecimale getal zou staan? Vast niet.

Wilt u een hexadecimaal getal onder BASIC omzetten in een decimaal getal dan, doet u dat op de volgende manier;

```
PRINT &HFF  ==>>  255
```

Wilt u een decimaal getal onder BASIC

omzetten in een hexadecimaal getal, dan doet u dat op de volgende manier;

```
PRINT HEX$(255) ==>> FF
```

N.B. In dit boek vindt u alle hexadecimale getallen voorzien van een 'H' achter het getal. Dit is gedaan, om het u zo makkelijk mogelijk te maken bij het schrijven van uw programma's in assemblertaal.

Het verband tussen binair en hexadecimaal

Een byte is opgebouwd uit 8 bits (nullen en eenen). Deze byte is te verdelen in twee 'nibbles' (een halve byte). Zo'n nibble kan een maximale waarde van 15 (FH) aannemen. Een binair getal wordt dan ook omgezet in een hexadecimaal getal door elk van de nibbles in HEX. vorm te noteren en achter elkaar te schrijven.

Voorbeeld: 11111010 binair

Bestaat uit twee nibbles, nl.: 1111 & 1010
Het eerste nibble is gelijk aan 15 = OFH
Het tweede nibble is gelijk aan 10 = OAH

Het getal 11111010B is dus gelijk aan FAH.

Het octale talstelsel

Het octale talsysteem wordt bij de MSX range van computers vrijwel niet gebruikt.

Het grondtal is 8.

Het systeem werkt als volgt;

512	64	8	1
8^3	8^2	8^1	8^0

U kunt decimale getallen omzetten in octale getallen d.m.v. de functie OCT$.

```
PRINT OCT$(500) ==>> 764 octaal

PRINT &o764    ==>>  500 decimaal
```

Tot zover ons verhaal over getallen. U gaat nu kennis maken met enkele begrippen betreffende het geheugen.

Het geheugen

Bij de bespreking van het geheugen gaan we er vanuit dat u de beschikking heeft over een 64K MSX computer.

Het geheugen van een 64K MSX computer bestaat uit 64 kilobytes. Een kilobyte bestaat uit 1024 bytes (geen 1000!). Een byte komt overeen met een geheugenplaats. Dat betekent 64*1024=65536 geheugenplaatsen voor ons gebruik.

Geheugenadressen kunnen bepaalde waarden bevatten. Als u in BASIC programmmeert, hebben alle bytes van 0 t/m 32768 (0-8000H) een vaste waarde. Dit gebied van het geheugen is nl. bezet door het ROM. Boven de 32768 is het RAM geheugen voor ons ter beschikking. De bytes in dit gebied kunnen verschillende waarden aannemen, afhankelijk van de programmatuur die zich in het RAM geheugen bevindt. Het bovenste deel van het RAM geheugen wordt gebruikt door uw computer om enkele belangrijke gegevens op te slaan (de zogenaamde systeemvariabelen). Enkele van deze gegevens zijn: regelbreedte, schermkleur etc.

Instrukties om in BASIC met de inhoud van geheugenadressen te kunnen manipuleren zijn peek & poke. Met PEEK kunnen we een bepaald adres uitlezen.

Voorbeeld: PRINT PEEK(40000) (ENTER)

Geeft als resultaat de inhoud van geheugenplaatsnr. 40000
Met de instruktie POKE kunnen we een bepaald geheugenadres vullen met een waarde.

Voorbeeld: POKE 40000,20 (ENTER)

Plaatst de waarde 20 in geheugenplaats nr. 40000.

Tik nu in POKE 45000,22 (ENTER)

Hierna PRINT PEEK(45000) (ENTER)

Als resultaat ziet u nu inderdaad het getal 22 omdat we allereerst geheugenadres nr. 45000 hebben gevuld met het getal 22 en daarna met een PEEK opdracht dit zelfde adres hebben uitgelezen.

Probeer eens het volgende...

10 REM MSX MACHINECODE (ENTER)

POKE 32773,130 (ENTER)

LIST (ENTER)

REM is veranderd in FOR omdat de computer op het adres 32773 de token (= een code voor een BASIC statement die door de computer wordt gehanteerd) van het eerste BASIC woord verwachtte en we dit d.m.v.

een POKE instruktie hebben veranderd van de code voor REM in de tokencode voor FOR. Meer over deze tokens wat later.

Het acht bits probleem

Zoals u weet is het bij het gebruik van 8 bits getallen niet mogelijk om waarden groter dan 255 te vormen. Hoe moet dat dan, als u een getal als 30300 wilt opslaan in het geheugen. We moeten dit getal dan allereerst delen door 256. De uitkomst van deze deling ronden we af naar het dichtstbij liggende gehele getal.

30300/256 = 118.36

Afgerond levert ons dit het getal 118. Dit is het hoge deel van het getal dat we willen opslaan. Dit hoge deel vermenigvuldigen we nu met 256.

118*256 = 30208

Dit antwoord (30208) trekken we nu af van het getal dat we willen opslaan.

30300-30208 = 92

Dit antwoord (92) is het z.g. lage deel van het getal dat we willen opslaan.

We zijn nu zo ver dat we het getal 30300 kunnen opslaan. Allereerst wordt het lage deel en dan het hoge deel in het geheugen geplaatst. Stel dat we het getal wilden opslaan op adres 8020H. Dit gebeurt dan op de volgende wijze:

```
POKE &H8020,92  (ENTER)
POKE &H8021,118 (ENTER)
```

We hebben dus twee geheugenplaatsen nodig om een getal, groter dan 255, op te slaan.

Om een getal uit te lezen dat groter is dan 255 gebruiken we de tegenovergestelde manier van het invoeren.

Stel u wilt het startadres van een machinetaal programma weten. Dit ligt normalerwijs tussen de 8000H en de F500H. Dat zijn dus getallen die groter zijn dan 255. Het startadres ligt vast in de top van het RAM geheugen. De adressen die dit start adres bevatten zijn: FCBFH & FCC0H.

Het opvragen van het startadres gebeurt dus als volgt:

```
PRINT HEX$((PEEK(&HFCBF))+256*(PEEK(&HFCC0)))
```

Het RAM geheugen

We hebben al eerder aangegeven, hoe het RAM is ingedeeld. We kwamen toen ook op de zogenaamde systeemvariabelen. Deze systeem variabelen bevinden zich vanaf geheugenplaats F380H. Deze systeemvariabelen worden door het ROM gebruikt om informatie op te halen over de door u geprogrammeerde functies.

Voorbeeld: U geeft een instruktie in BASIC om de regelbreedte in screenmode 0 in te stellen d.m.v. WIDTH 30
Hierdoor zal schermmode 0 een maximaal aantal karakters van 30 op een regel plaatsen.

De informatie over de schermbreedte in schermmode 0 staat op F3AEH. F3AEH bevat de waarde van het aantal karakters in screenmode 0. U zou dus net zo goed d.m.v. een poke instruktie het aantal karakters op het scherm kunnen instellen. U moet dan echter na de poke instruktie ook de juiste scherm mode instellen (in dit geval screen0).

```
Voorbeeld: POKE &HF3AEH,20 (ENTER)
           SCREEN0 (ENTER)
```

Het ROM geheugen

ROM staat voor Read Only Memory. Dit betekent dat het een geheugen is waaruit alleen kan worden gelezen. Het ROM is verdeeld in twee delen.

0000H - 3FFFH	Operating System
4000H - 7FFFH	BASIC

De ROM bevat zeer veel nuttige informatie voor de machinetaal programmeur. De inhoud van de ROM is nl. niets anders dan een groot blok machinetaal. Deze machinetaal kan d.m.v. speciale opdrachten door u worden gebruikt in uw eigen programma's. Dit wordt gedaan d.m.v. de opdracht CALL in machinetaal of een DEFUSR=ROMADRES: X=USR(0) in BASIC.

In appendix vindt u de adressen van de meest zinvolle ROM-adressen.

Het Video RAM

Uw MSX computer is uitgerust met de TMS 9918A video processor. Deze chip heeft een eigen 16K RAM. In BASIC kunt u d.m.v. van de instrukties VPEEK en VPOKE gebruik maken van dit Video RAM.

GEHEUGEN BANKEN OVERZICHT

De MSX computers zijn maximaal in staat om 64K RAM tegelijkertijd te gebruiken. Het is echter mogelijk om d.m.v. van z.g. RAM-expansion interfaces het geheugen uit te breiden tot een theoretisch maximum van 1024 Kbytes... Dit is echter in de praktijk vrijwel onmogelijk. Een MSX computer is nl. uitgerust met 4 slots. Elk van deze slots is weer uit te breiden tot maximaal 4 slots. Zo kunnen er maximaal 16 slots worden gevormd. Deze slots kunnen in principe allemaal met RAM worden gevuld. Dit is dan maximaal 64K per slot. Dat is dus 64*16=1024K RAM. Daar echter enkele sloten niet zomaar zijn uit te breiden zal in de praktijk een geheugenuitbreiding tot ongeveer 256K RAM tot de mogelijkheden behoren. Deze extra RAM is door de beperking van de Z80 slechts via een methode, die bank-switching wordt genoemd, aan te spreken. Hoe dit in z'n werk gaat zullen we u later uit de doeken doen.

Nu is het tijd om u een visueel overzicht te geven van een mogelijke geheugenindeling van een MSX computer.

	SLOT 0	SLOT 1	SLOT 2	SLOT 3
0000H pagina 0	BASIC ROM	RAM	CARTRIDGE SLOT	
4000H pagina 1	BASIC ROM	RAM		DISK ROM
8000H pagina 2		RAM		
C000H pagina 3		RAM		
FFFFH				

Geheugen indeling onder Basic

	Begin van een programma (8000H/TEXTAB=8001H)
BASIC PROGRAMMA	
	Start variabelen tabel (VARTAB=8003H)
VARIABELEN TABEL	
	Start array tabel (ARYTAB=8003H)
ARRAY TABEL	
	Einde array tabel (STREND=8003H)
WERK RUIMTE	
	Basic stack pointer
BASIC STACK	
	(STKTOP=F0A0H) Einde van de string ruimte String pointer Begin van de string ruimte (FRETOP=F168H/MEMSIZ=F168H)
STRING SPACE	
I/O BUFFERS	Top van het BASIC geheugen NULBUF=F177H
MACHINE- TAAL & DISK BASIC	HIMEM=F380H
SYSTEEM VARIABELEN	F380H-FD99H

Het geheugen onder Basic

Het startadres van een BASIC programma ligt altijd op 8000H.

De lengte van een BASIC programma is variabel. Het einde ligt aan het begin van de variabelen tabel.

Het adres van de variabelen tabel: F6C2H

DE BASIC IN WERKING

Een BASIC programma wordt in binaire vorm opgeslagen in het geheugen. BASIC statements worden opgeslagen als codes die een lengte hebben van 1 of 2 bytes. Deze codes noemt men tokens. Nummers worden opgeslagen in binaire vorm en letters als ASCII codes. Een voorbeeld van deze tokens heeft u reeds kunnen zien.

Naast de tokens, worden er regelnummers opgeslagen. Lijnen worden op de volgende wijze opgeslagen;

De eerste twee bytes bevatten de locatie in het geheugen waar de volgende regel BASIC begint. Deze twee bytes zijn gereserveerd. De volgende twee bytes bevatten het huidige regelnummer. Twee bytes zijn nodig om regelnummers tot 65535 te kunnen laten oplopen, er wordt echter slechts 1 byte gebruikt bij regelnummers kleiner dan 256. Deze bytes zijn ook gereserveerd. Nu

is er een regel 'token-BASIC' opgeslagen, gevolgd door een byte die op nul staat. (Dat geeft het einde van de regel aan)

Hieronder volgt een voorbeeld van BASIC tokens.

```
400 deffna(x)=cos(x)
500 print fna(1)
```

Elk programma begint normalerwijs op 8001H.

Het geheugen ziet er als volgt uit....

loc	8001H	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10
HEX.	12	80	90	01	97	DE	41	28	58	29	EF

loc	8001H	+11	+12	+13	+14	+15	+16
		FF	8C	28	58	29	00

loc	8012H	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10
HEX.	1E	80	F4	01	91	20	DE	41	28	12	29

loc	8012H	+11	+12	+13	+14	+15	+16
		00	00	00	08	C1	00

Adressen 8001H en 8002H vormen de locatie van de volgende regel in het BASIC programma. In dit geval 1280, of als we de bytes omdraaien, 8012H.
Dan twee bytes met het huidige regelnummer, 9001H, ofwel 0190H (400 decimaal). De BASIC codes komen hierna. Locatie 8001H +4 bevat de waarde 97, dit is de tokenwaarde voor DEF, en de volgende byte (DE), is de token voor FN. De volgende bytes

zijn geen tokens. De ASCII code voor deze waarden is 'a(x)', (41,28,58,29). De volgende byte is EF, de token voor het '='-teken. FF en 8C is de token voor COS. Hierna weer enkele ASCII codes voor '(x)', gevolgd door een nul aan het eind van de regel.

We hebben kunnen zien dat het begin van de volgende regel met BASIC codes ligt op adres 8012H. De twee bytes 8012H en 8013H bevatten de pointer naar de volgende regel. In dit geval 801E. Dan F401 (eigenlijk 01F4) is gelijk aan 500 decimaal onze huidige regel. De volgende byte, 91, is de token voor PRINT, hierna 20, dat voor de code van een spatie staat. Nu komen we op de token van FN gevolgd door a(1). De volgende 0 is de afsluiting van de regel. De nul die daarop volgt is de 0 die aangeeft dat het programma is afgelopen.

Hopelijk heeft u nu een iets beter inzicht in hoe de computer een BASIC programma in het geheugen opslaat. Experimenteer rustig met deze kennis, er zijn hele leuke dingen mee te doen.

Hieronder vindt u de lijst van beschikbare BASIC woorden en hun tokens. (De tokentabel begint op adres 3A72H in het ROM)

AUTO	A9H	AND	F6H
ABS	06H	ATN	0EH
ASC	15H	ATTR$	E9H
BASE	C9H	BSAVE	D0H
BLOAD	CFH	BEEP	C0H
BIN$	1DH	CALL	CAH

CLOSE	B4H	COPY	D6H
CONT	99H	CLEAR	92H
CLOAD	9BH	CSAVE	9AH
CSRLIN	E8H	CINT	1EH
CSNG	1FH	CDBL	20F
CVI	28H	CVD	2AH
COS	0CH	CHR$	16H
CIRCLE	BCH	COLOR	BDH
CLS	9FH	CMD	D7H
DELETE	A8H	DATA	84H
DIM	86H	DEFSTR	ABH
DEFINT	ACH	DEFSNG	ADH
DEFDBL	AEH	DSKO$	D1H
DEF	97H	DSKI$	EAH
DSKF	26H	DRAW	BEH
ELSE	A1H	END	81H
ERASE	A5H	ERROR	A6H
ERL	E1H	ERR	E2H
EXP	0BH	EOF	2BH
EQV	F9H	FOR	82H
FIELD	B1H	FILES	B7H
FN	DEH	FRE	0FH
FIX	21H	FPOS	27H
GOTO	89H	GO TO	89H
GOSUB	8DH	GET	B2H
HEX$	1BH	INPUT	85H
IF	8BH	INSTR	E5H
INT	05H	INP	10H
IMP	FAH	INKEY$	ECH
IPL	D5H	KILL	D4H
KEY	CCH	KEY	CCH
LPRINT	9DH	LLIST	9EH
LPOS	1CH	LET	88H
LOCATE	D8H	LINE	AFH
LOAD	B5H	LSET	B8H
LIST	93H	LFILES	BBH
LOG	0AH	LOC	2CH
LEN	12H	LEFT$	01H
LOF	2DH	MOTOR	CEH

MERGE	B6H
MKI$	2EH
MKD$	30H
MAX	CDH
NAME	D3H
NOT	E0H
OUT	9CH
OR	F7H
OFF	EBH
PUT	B3H
POS	11H
PSET	C2H
POINT	EDH
PDL	24H
PLAY	C1H
READ	87H
RESTORE	8CH
RESUME	A7H
RIGHT$	02H
RENUM	AAH
SPRITE	C7H
SWAP	A4H
SAVE	BAH
STEP	DCH
SQR	07H
STR$	13H
SPACE$	E3H
STICK	22H
THEN	DAH
TROFF	A3H
TO	D9H
TAN	0DH
USR	DDH
VARPTR	E7H
VPOKE	C6H
WIDTH	A0H
XOR	F8H
MOD	FBH
MKS$	2FH
MID$	03H
NEXT	83H
NEW	94H
OPEN	B0H
ON	95H
OCT$	1AH
PRINT	91H
POKE	98H
PEEK	17H
PRESET	C3H
PAINT	BFH
PAD	25H
RETURN	8EH
RUN	8AH
REM	8FH
RSET	B9H
RND	08H
SCREEN	C5H
STOP	90H
SET	D2H
SPC(	DFH
SGN	04H
SIN	09H
STRING$	E3H
SOUND	C4H
STRIG	23H
TRON	A2H
TAB(	DBH
TIME	CBH
USING	E4H
VAL	14H
VDP	C8H
VPEEK	18H
WAIT	96H

Adres 3D26H in het ROM bevat de token-tabel voor afzonderlijke karakters.

+	F1H	-	F2H	^	F5H	'	E6H	=	EFH
*	F3H	/	F4H	>	FCH	<	F0H	\	FCH

Variabelen

Variabelen zijn kleine hokjes in het geheugen. Er bestaan twee soorten variabelen, zij die tekst opslaan (string variabelen) en zij die getallen opslaan (numerieke variabelen). Een variabele in een BASIC programma moet bestaan uit 1 of 2 karakters (bijvoorbeeld AA of X).
Bijvoorbeeld:

```
LET XX=300.550 OF X=10000  ==>> Numeriek

LET A$="TEST" of A$="TEST" ==>> String
```

Als u variabelen definieert die groter zijn dan twee karakters, dan let de computer alleen op de eerste twee karakters.

Uw MSX heeft alle informatie over variabelen in de variabelen tabel.

Variabelen tabel

De variabelen tabel bevat informatie over de gebruikte variabelen in een BASIC programma. De waarde van nummerieke variabelen kunt u hierin terugvinden. Van de stringvariabelen kunt u hier alleen de lengte en het adres vinden.

Het variabelentabel-startadres bevindt zich op adres F6C2H.

Voorbeeld van het gebruik van de variabelen tabel:

TIK IN: 10 a$="DIT IS EEN STRING"
RUN (ENTER)

Het tabel-startadres vindt u nu door in te tikken:

```
PRINT HEX$(PEEK(&HF6C2)+256*PEEK(&HF6C3))
(ENTER)
```

Nu tikt u in:

```
FOR X=&H801E TO &H8023:?PEEK(X)" ";:NEXT
(ENTER)
```

3 65 0 17 9 128

Wat houden nu al deze getallen in ?

801EH 3 - Is de code voor een stringvariabele.
801FH 65 - Is de ASCII waarde van 'A'.

8020H	00	- We hebben een string van slechts 1 letter gebruikt. Daar een MSX computer naar 2 karakters 'kijkt', wordt deze als een leeg karakter beschouwd.
8021H	17	- Is het aantal karakters dat zich in de string bevindt.
8022H	09	- De 9 en de 128 vormen samen een adres. (09+256*128=8009H)
8023H	128	- Tik in PRINT(PEEK(&H8009)) (ENTER). Dit geeft als uitkomtst 68, wat overeen komt met de ASCII code voor de letter 'D' ,de eerste letter uit de string.

Een BASIC statement dat tot doel heeft u meer informatie te verschaffen over de variabelen is 'VARPTR'. Dit statement werkt als volgt...

```
Tik in: A$="MSX" (ENTER)
        PRINT HEX$(VARPTR(A$)) (ENTER)
Resultaat: 8021H
```

Dit is het adres waar de informatie over de informatie over de string "MSX" ligt opgeslagen.

```
Tik nu in: PRINT PEEK(&H8021) (ENTER)
Resultaat: 3
```

Drie is de lengte van de string "MSX"

Experimenteer wat met het statement VARPTR, hierdoor krijgt u een goed inzicht hoe uw computer variabelen behandelt.

Basic stapel

De BASIC STAPEL bevindt zich in het geheugen tussen het einde van de STRING ruimte en de STAPEL-POINTER. De stapel pointer geeft het einde aan van de stapel. Deze groeit naar beneden vanaf het einde van de string ruimte.

Dit betekent dat de top van de stapel zich altijd op het laagst mogelijke adres in het geheugen bevindt.

De stapel wordt door de BASIC gebruikt om ervoor te zorgen, dat de commando's GOSUB en FOR-NEXT goed werken. Na een GOSUB is het voor de computer wel zo makkelijk dat deze weet waar hij is gebleven. Als een MSX computer nu een GOSUB tegenkomt, dan zet hij een waarde op de STAPEL die overeenkomt met de plaatst van de GOSUB opdracht in het programma. Als de MSX aan het eind van de subroutine een RETURN tegenkomt, dan pakt hij de laatste waarde van de STAPEL en verhoogt deze met 1. Nu loopt het programma gewoon verder, zonder problemen.

Als de STAPEL verkeerd wordt gebruikt zal het geheugen zeer snel vol raken.
Probeer maar eens;

```
100 GOSUB 100  (ENTER)
           RUN  (ENTER)
```

De boodschap 'OUT OF MEMORY' zal binnen zeer korte tijd op het scherm zichtbaar worden.
De verklaring voor dit fenomeen ligt in het feit dat bij elke gosub een 7 byte groot adres op de STAPEL wordt geplaatst. Dit adres wordt alleen gewist als er een RETURN wordt tegengekomen. Aangezien wij in ons programma geen return hebben staan zal de STAPEL zeer snel het geheugen vullen.

Een FOR-NEXT loop zet 25 bytes op de STAPEL. Deze bytes worden daar alleen vanaf gehaald, als alle stappen zijn doorlopen. Als u door een voorwaarde uit een FOR-NEXT lus springt, dan zal dit tot gevolg hebben dat er 25 bytes op de STAPEL blijven staan.

Om problemen te voorkomen, kunt u het best alle FOR-NEXT lussen in subroutines verwerken. De RETURN veegt zowel de FOR-NEXT bytes alswel de GOSUB bytes van de STAPEL.

Tot zover onze inleiding in het MSX computer gebeuren.

Wat wij niet hebben behandeld in deze inleiding, zijn de Video Display Processor, met zijn Video Ram en registers. De gegevens hierover heb ik verwerkt in een boek, getiteld 'MSX verder uitgediept - STARK'. Dit boek gaat ook in op de systeem routines, ROM routines, Sprites etc. etc. Er

staat ook een programma in om het BEGIN-, EIND- en STARTADRES van een machinetaal programma op te vragen. Dit programma is zeer makkelijk bij het overzetten van cassette software naar DISK. Vraag naar dit boek bij uw handelaar.

De microprocessor

MSX computers worden door de concurentie vaak omschreven als verouderd. Als belangrijkste reden voor deze uitspraak wordt aangevoerd dat het gebruikte 'hart' van deze computer niet meer aan de hedendaagse eisen zou voldoen.
Het hart van uw computer wordt nl. gevormd door de Z80A microprocessor. Deze processor is inderdaad al enige jaren op de markt. Dat de processor verouderd is, ben ik niet helemaal eens met de MSX-critici. De Z80 is een van de krachtigste, zo niet de krachtigste 8 bits processor. Er zijn zelfs mogelijkheden om met 16 bits getallen te werken. Dit in tegenstelling tot bijvoorbeeld tot de toch ook veel gebruikte 6502 microprocessor. De snelheid van de Z80A bedraagt 4 Mhz. Dit is ongeveer twee tot vier maal zo snel als de gemiddelde homecomputer.
Het grootste voordeel, is naar mijn mening de CP/M standaard, die is ontwikkeld voor de Z80 microprocessor. Daar het voor MSX ontwikkelde DOS compatible is met CP/M is er voor uw computer na de aanschaf van een diskdrive zeer veel goede programmatuur beschikbaar.
Naar mijn idee zijn een goede Video- en Sound Chip veel belangrijker voor een homecomputer.

Na deze beschouwing van de eigenschappen en hun voor/nadelen kunnen we nu de Z80

aan een nauwere inspectie onderwerpen.

De Z80 heeft als taak om ervoor te zorgen dat alles wat in de computer gebeurt, te besturen. Hiervoor is de processor uitgerust met enkele registers. Deze registers kunnen we gebruiken voor datamanipulatie. De registers zijn vrijwel allemaal 8 bits. Maar door de combinatie van twee 8 bits registers is het mogelijk om een 16 bits register te vormen. De processor kan de bytes uitwisselen met bytes in het RAM geheugen. De Z80 bestuurt maximaal 64K aan geheugen.

Wat het rekenwerk betreft is de processor niet echt intelligent te noemen. De Z80 is slechts in staat om optellingen en aftrekkingen te maken. Vermenigvuldigen en delen gaat op een vrij omslachtige wijze. Maar omdat een handeling zo ontzettend snel verloopt is van enig tijdverlies bijna geen sprake.

De basis van alle handelingen ligt in logica. Mensen met een wiskunde achtergrond zullen waarschijnlijk ook wat minder moeite hebben met de werkingswijze van de Z80.

Naast de rekenkundige handelingen is de processor ook in staat om enkele logische operaties uit te voeren. Deze logische operaties bestaan uit AND-, OR- en XOR-operaties. Logische operaties vergelijken twee bytes en produceren hieruit een antwoord. De AND operatie heeft als resultaat

een 1 als beide bits gelijk zijn aan 1 anders 0. In BASIC kunnen we het AND statement bijvoorbeeld gebruiken bij een voorwaarde.

Voorbeeld:

```
IF A=10 AND B$="MSX" THEN PRINT......
```

BASIC geeft als uitkomst 11111111 als A=10 en B$ gelijk is aan MSX. Als dit niet het geval is, is de uitkomst gelijk aan 00000000.

OR geeft als resultaat 11111111 als een van de twee bits gelijk is aan 1 of als ze beide gelijk zijn aan 1. Zijn ze allebei gelijk aan 0 dan is het resultaat 00000000.

Voorbeeld in BASIC:

```
IF A=10 OR B$="MSX" THEN ......
```

XOR is de eXclusieve OR. Dit is een OR opdracht met als uitzondering dat als beide bits gelijk zijn aan 1 de uitkomst ook gelijk is aan 0.

Hieronder vindt U een overzicht van de logische operaties. We voeren hiervoor twee variabelen in (twee bits A en B), en een antwoordvariabele (bit) in de vorm van S.

Logische operaties

AND

Operatie: A and B = S

A	B	!	S
0	0	!	0
0	1	!	0
1	0	!	0
1	1	!	1

```
Voorbeeld:  01101111
            11110110 AND
            --------
            01100110
```

OR

Operatie: A or B = S

A	B	!	S
0	0	!	0
1	0	!	1
0	1	!	1
1	1	!	1

```
Voorbeeld:  01100111
            01110110 OR
            --------
            01110111
```

XOR

Operatie: A XOR B = S

```
A B ! S
-------
0 0 ! 0
1 0 ! 1
0 1 ! 1
1 1 ! 0
```

```
Voorbeeld:  01011111
            11000110   XOR
            --------
            10011001
```

OPDRACHT: Voer een XOR instruktie uit op de BYTE-a, 01010101 met de BYTE-b, 11111111.

Het resultaat zal zijn dat de inverse van BYTE-a als resultaat op het scherm zal verschijnen.

Een praktische toepassing van dit inverteren van bytes vindt u hieronder. Hier wordt een gebrek van MSX (geen inverse karakters) d.m.v. XOR verholpen. Alle karakters zijn opgebouwd uit een groep van 8 bytes (zoals een 8*8 sprite). Door met elk van deze bytes een XOR-operatie uit te voeren (met 11111111), verkrijgen we een

inverse karakter set. De inverse karakterset zetten we daarna achter de normale in het Video Ram.

```
10 SCREEN 0
20 FOR I=2304 TO 3055
30 A=VPEEK(I)
40 A=A XOR &B11111111
50 VPOKE I + 768,A
60 NEXT
```

(RUN)

UITLEG: 10 Het tekstscherm.
20 Locatie in Video RAM waar zich de MSX karakterset bevindt.
30 inlezen van de bytes.
40 De bytes worden geinverteerd.
50 De geinverteerde bytes worden 768 posities verder in het Video RAM geplaatst.

TER CONTROLE:

Tik in:FOR X=1TO255:?CHR$(X);:NEXT (ENTER)

Binair rekenen

Het rekenen met binaire getallen is niet zo moeilijk als het er op het eerste gezicht uitziet. Het is echter wel een belangrijk onderdeel bij het programmeren in machinetaal. We zullen beginnen met het optellen en aftrekken. Daarna het wat moeilijkere vermenigvuldigen.

Binair optellen:

We zullen de getallen 15-binair en 6-binair gaan optellen. 15-binair komt overeen met 1111 en 6-binair met 0110.

```
    15    1111
    06    0110 +
          ----
    21   10101
```

We zullen nu in een schema bekijken hoe twee bits bij elkaar worden opgeteld. Hiervoor voeren we de variabelen A en B in voor de op te tellen bits. Het resultaat komt in S, en een eventuele carry in C. Een carry is een rest, dat ontstaat nadat twee bits met de waarde 1, bij elkaar worden opgeteld.

```
A B ! S C
---------
0 0 ! 0 0
1 0 ! 1 0
0 1 ! 1 0
1 1 ! 0 1
```

Het binair aftrekken lijkt sterk op het decimaal aftrekken. We zullen ook nu weer een voorbeeld geven, echter nu beginnen we met een decimale aftreksom.

```
842
459 -
---
383
```

Een nadere uitleg van bovenstaande zal u misschien wat schools overkomen, maar de ervaring leert dat zoiets nooit kwaad kan. Als we de 9 van de 2 willen aftrekken, moeten we een tiental lenen van 4, omdat we anders een negatief getal als resultaat krijgen. Nu ontstaat de som 12-9=3. Nu willen we 5 van de 3 aftrekken (3=4-1). We lenen deze keer van de 8. Nu ontstaat de som 13-5=8. Nu hebben als laatste de som.
7-4=3. Als uiteindelijk resultaat houden we dus over: 383.

We zullen nu een voorbeeld zien van een binaire aftreksom.

```
10010
01101 -
-----
00101
```

We trekken allereerst 1 van 0 af. Dit geeft een negatief resultaat. We lenen nu een tweetal van de 1. Nu ontstaat de aftreksom 2-1=1. Nu trekken we 0 van 0 af met als resultaat een 0. Nu moeten we een 1 van 0 aftrekken. Het resultaat hiervan is negatief. We moeten dus lenen. Dat kan niet bij een 0, dus moeten we het een bit verder proberen.
Hier staat een 1, dus dat zou kunnen. We lenen een tweetal. Dat brengen we over naar de eerste 0. Nu er een tweetal staat kunnen we hiervan een ander tweetal lenen. Als resultaat ontstaat hieruit de volgende binaire aftreksom:

```
012
011-
---
001
```

Als uiteindelijk resultaat krijgen we dus 00101.

Tot zover het binaire optellen en aftrekken.

Om te oefenen hieronder enkele opgaven:

```
A)          B)           C)             D)
01101       101010       0100111        1101011
00111 -     010101 -     1100101 +      0111011 +
-----       ------       -------        -------
```

De antwoorden op de opgaven zijn:

A: 00110
B: 010101

C: 10001100
D: 10100110

Naast de standaard regels van het optellen en aftrekken bestaan er ook die van de tweecomplementen.

Voordat we gaan kijken wat een tweecomplement in houdt, zullen we eerst laten zien waar we het eigenlijk voor nodig zullen hebben.

Bij sommige aftrek sommen, zal de hiervoor gebruikte methode zeer snel tot fouten lijden. Bij de tweecomplements methode zijn deze fouten geheel uitgesloten, omdat dan alle aftreksommen worden teruggevoerd tot eenvoudige optellingen.

```
10101111 = 175
01110110 = 118
-------- -
00111001 =  57
```

Het bovenstaand probleem kunnen we vereenvoudigen als we gebruik maken van tweecomplementsgetallen.

Tweecomplements getallen

Positieve binaire getallen blijven onveranderd bij het tweecomplements rekenen. (Voorbeeld: 01110100)

Een negatief getal echter, wordt wel veranderd. Het negatieve getal moet ten eerste worden geinverteerd. (Alle eenen worden 0 en alle nullen worden 1) Hierna moet bij dit geinverteerde getal 1 worden opgeteld.

Voorbeeld: 00000011 (3 dec.)
Complement: 11111100

Het meest significante bit (nr.7) is het z.g. teken-bit. Als dit bit is geset (1) dan is het getal negatief, anders is het positief (bitnr.7=0).

P.S. Het complement is gelijk aan de inverse van een getal. (niet het 2-complement!)

Nu bepalen we het tweecomplement door 1 bij het complement op te tellen;

```
11111100
00000001
-------- +
11111101
```

We hebben nu het getal -3 in tweecomplements uitvoering.

Als u de waarde van een tweecomplements getal wilt weten dan neemt u van dat getal weer het tweecomplement en als resultaat zal dan het positieve getal ontstaan.

```
Voorbeeld:      11111101   (2-complement)
                00000010   (geinverteerd)

                00000010
                00000001
                -------- +
                00000011   (3 decimaal)
```

We zullen nu een voorbeeld zien van het gebruik van tweecomplementen. Hiervoor nemen we de al eerder behandelde binaire-aftreksom.
Nu zullen we dit probleem nog eens aanpakken, maar nu met de tweecomplementsregels.....

```
getal-a:  10101111  - 175
getal-b:  10001110  - 118
          -------- -
```

We nemen nu van getal-b het tweecompement.

```
inverse:  10001001
          00000001
          -------- +
getal-c:  10001010
```

Getal-c is een negatief getal. Dit getal gaan we nu optellen bij getal-a. Hierdoor ontstaat een tweecomplement-aftreksom.

```
           10101111
           10001010
           -------- +
getal-d:  100111001
```

Het getal-d is nu de uitkomst van onze aftreksom. Het verschil met ons eerdere antwoord is het bitnr.-8. Dit bit staat nu op 1 i.p.v. 0. Dit komt door de inversebepaling. Haalt u dit bit weg, dan zal het goede antwoord ontstaan.
n.l.: 00111001 = 57 = 37H

Binair vermenigvuldigen

Om deze handelingen in machinetaal te kunnen uitvoeren, heeft u de zogenaamde schuifinstrukties nodig. Deze instrukties verschuiven een byte in z'n geheel naar rechts of naar links. Dit kan gebeuren met of zonder carry. Daar we op dit moment nog niet aan de bespreking toe zijn zullen we deze ook nu nog niet aan de orde brengen.

Bij de bespreking van de schuifinstrukties zullen we een voorbeeld van het vermenigvuldigen in machinetaal geven in de vorm van een korte routine.

Hieronder volgt een uitleg van het vermenigvuldigen in binaire vorm. Allereerst een vermenigvuldiging in het decimale systeem.

```
  30
 223 *
 ---
  90
 600
6000 +
----
6690
```

Bovenstaand voorbeeld zult u zeker begrijpen. We zullen nu dezelfde vermenigvuldiging in binaire vorm gaan uitvoeren.

```
 30 = 1EH = 00011110
223 = DFH = 11011111


          11011111
          00011110 *
          --------
          00000000
         110111110
        1101111100
       11011111000
      110111110000
     0000000000000
    00000000000000
   000000000000000 +
   ---------------
  0001101000100010
```

Het binair vermenigvuldigen op zich is eigenlijk niet zo moeilijk. De grote optellingen zijn echter vaak niet erg overzichtelijk. Hierdoor ontstaan meestal de fouten.

Wat is machinetaal?

Het hart van de huidige MSX computers is zoals u reeds weet, de Z80A microprocessor. Alle opdrachten aan de computer verlopen via deze processor.
Als u onder BASIC werkt, dan hoeft u zich geen zorgen te maken over hetgeen de processor moet uivoeren, de BASIC-interpreter zorgt hiervoor. Als u een commando onder BASIC geeft, dan wordt dit door de interpreter omgezet in een taal die de Z80A begrijpen kan: de Z80 machine taal.
Echter als u een programma in machinetaal schrijft, dan moet u de processor zelf opdrachten geven die hij begrijpt.

Waarom machinetaal leren als er een BASIC interpreter in de computer zit?
Het antwoord op deze vraag hebben we in het begin van het boek reeds gegeven, maar laten we het nog maar eens herhalen...

Ten eerste kan men onder BASIC niet het uiterste uit de computer halen. Zo bestaat er geen BASIC routine om het beeld te laten scrollen.

Ten tweede een machinetaal routine is veel sneller dan een BASIC routine, dit scheelt soms een factor honderd. Hieronder volgt een voorbeeld om het verschil in snelheid aan te tonen. Dit programma is slechts ter illustratie v/d snelheid van machinetaal. Later zult ook u begrijpen hoe het pro-

gramma werkt.

Programma 1 is in BASIC geschreven, programma 2 in machinetaal. Beide programma's maken gebruik van de ingebouwde klok zodat u de snelheden kunt vergelijken.

Programma 1 (BASIC PROGRAMMA)

```
10 SCREEN0:WIDTH40:CLS:KEYOFF
20 DEFINT T:TIME=0
30 FOR T=0 TO 959
40 PRINT "p";:NEXT:LOCATE 1,1
50 PRINTTIME
```

Programma 2 (MACHINETAAL PROGRAMMA)

Dit programma valt uiteen in twee delen, deel a is de benodigde machinetaal en deel b voert het programma uit. Tik eerst deel a in en 'run' het.
'run' vervolgens deel b.

Deel a:

```
10 DATA 62,112,33,0,0,1,192,3,205,86
20 DATA 0,201
25 CLEAR 200,&HAFFF
30 FOR T=&HB000 TO &HB00B
40 READ A:POKE T,A:NEXT
50 DEFUSR=&HB000
60 NEW
```

deel b:

```
10 CLS:TIME=0:X=USR(0):PRINT TIME
20 END
```

De Z80 machinetaal

De Z80 microprocessor heeft de beschikking over een aantal registers, dit zijn een soort geheugens die hij nodig heeft om z'n opdrachten uit te voeren. Deze registers kunnen door de programmeur veranderd worden. Daarvoor zijn instrukties nodig die de processor 'begrijpt '. Het vervelende is echter dat hij alleen maar elektrische signalen kan begrijpen. De processor kent in principe twee soorten signalen: Het signaal 'aan', er loopt een elektrische stroom en het signaal 'uit', er loopt geen elektrische stroom. M.b.v het binaire talstelsel kunt u deze signalen opwekken, een 1 betekent 'aan', een 0 betekent 'uit'.
De Z80 kan acht van deze signalen tegelijkertijd ontvangen, de manier waarop deze acht signalen gecombineerd worden is bepalend voor de reactie van de processor.
Zo reageert de processor b.v. anders op de combinatie '01010101' dan op de combinatie '11111111' . De eerste groep van acht signalen (uit, aan, uit, aan, uit, aan, uit, aan) is anders opgebouwd dan de tweede groep(aan, aan, aan, aan, aan, aan, aan, aan).

Voor de mens is het veel te moeilijk om alle combinaties en de daar bij behorende reacties uit z'n hoofd te leren, of telkens weer in een boek op te zoeken. Daarom is er door de fabrikant van de Z80 een

speciale taal ontwikkeld, de Z80 assembler taal. Voordat we hierop in gaan, zullen we eerst eens gaan kijken naar de registers.

De Z80 maakt gebruik van verschillende soorten registers d.w.z. de registers worden voor verschillende doeleinden gebruikt.

Het A-register.

Dit register wordt ook wel de accumulator genoemd. Vrijwel alle rekenkundige bewerkingen zoals optellen en aftrekken kunnen via dit register gerealiseerd worden. Het is een acht bits register, d.w.z. het kan met acht bits geladen worden oftewel 1 byte.

De B en C registers.

Deze registers worden meestal als paar gebruikt. Ze doen vaak dienst als een teller. Het zijn twee acht bit registers, maar tezamen kunnen ze een zestien bit registerpaar vormen.

De D en E registers.

Ook deze registers worden vaak als paar gebruikt. Het zijn eveneens twee acht bit registers die tezamen een zestien bit registerpaar kunnen vormen.

De H en L registers.

Hiervoor geldt het zelfde als bij de D en E registers.

De programma counter: afgekort PC.

Dit is een register bestaande uit twee bytes. De PC bevat het adres van de volgende instruktie. Elke keer als er een instruktie uitgevoerd is, wordt dit register automatisch, afhankelijk van de gebruikte instruktie, veranderd zodat de Z80 'weet' waar hij de volgende instruktie kan vinden.

Naast deze registers kent de Z80 er nog een aantal, echter deze komen in de programma's vanzelf ter sprake.

En dan nu verder met de assembler taal.
Een voorbeeld om het verschil aan te duiden tussen de machinetaal en de assembler taal:

Om het A register te laden met het getal 00000001 , dient u de volgende groepen signalen aan de Z80 door te geven: 00111110 en 00000001.
Het eerst groepje signalen, 00111110, betekent voor de Z80 dat hij de accumulator moet laden met het getal dat onmiddellijk hierop volgt, in dit geval 00000001. Het groepje signalen 00111110 stelt een binair getal voor. Omgezet naar het hexadecimale talstelsel levert dat het getal 3E op. Het tweede groepje signalen 00000001 stelt ook een binair getal voor en levert 01 hexadecimaal op. Daar een MSX computer hexadecimale getallen herkent en ze omzet naar binaire getallen, kunnen we bovengenoemde groepjes signalen ook hexadecimaal invoeren.

In assembler taal hebben we niets te maken met de getallen, we geven alleen maar een instruktie op net als onder BASIC. Een speciaal programma, een zogenaamd assembler, vertaalt deze instrukties in machinetaal.

Om het A register met de waarde 01H te laden, dient de volgende instruktie ingevoerd te worden:

```
LD  A,01H.
```

Dit betekent: laad register A met de waarde 01H.

Nog een voorbeeld:

We willen register B laden met de waarde 05H.

In machinetaal zijn daarvoor de volgende getallen nodig 06H en 05H. 06H is binair geschreven 00000110 en betekent voor de Z80 'laadt register B met het eerst volgende getal dat ik tegenkom'. In dit geval is dat 05H.
In assembler taal heeft de instruktie 'LD B,05H' het zelfde effect.

Hieronder volgen nog 4 voorbeelden van assembler instrukties:

```
1 LD H,FFH     ; LAAD REGISTER H MET FFH
2 LD HL,3412H  ; LAAD REGISTERPAAR HL MET
                 3412H
3 LD H,34H     ; SPREEKT VOOR ZICH
4 LD L,12H     ; SPREEKT VOOR ZICH
```

Instruktie 3 en 4 samen, hebben het zelfde effect als instruktie 2 (in beide gevallen bevat het H register de waarde 34H en het L register de waarde 12H), echter instruktie 2 is sneller.

Genoeg theorie, we gaan verder met een programma.

Doel van het programma: we willen geheugen plaats 9000H laden met de waarde 20H.
Dit programma heeft hetzelfde effect als de BASIC instruktie poke &h9000,&h20

Het assembler programma:

```
0 ORG 0A000H
1 LD  A,020H
2 LD (09000H),A
3 RET
4 END
```

Uitleg:

De regelnummering:

In dit boek zijn alle assembler programma's voorzien van regelnummers, echter niet alle assemblers herkennen dit. Als uw assembler geen regelnummering kent, laat hem dan weg.

regel 0: ORG A000H

Dit is een nieuw commando. Het betekent voor de assembler dat hij vanaf adres A000H de machintaal moet schrijven. Dus vanaf adres A000H worden de gegevens voor de Z80 geladen.

regel 1:

LD A,20H

Register A wordt met de waarde 20H geladen.

regel 2:

LD (9000H),A

Dit is een nieuwe instruktie en betekent 'laad geheugen plaats 9000H met de inhoud van de accumulator'. In dit geval wordt geheugen 9000H geladen met 20H.

regel 3:

RET

Ook dit is een nieuwe instruktie 'RET' staat voor return. In dit geval gaan we terug naar BASIC.
U kunt dit programmaatje zien als een subroutine, als deze subroutine is uitgevoerd, dan moet er weer naar het hoofd programma worden terug gekeerd, in dit geval is dat de BASIC in de computer (in feite is de BASIC zelf ook een zeer groot programma, geschreven in machinetaal).

regel 5:

END

Het END statement betekent voor de assembler dat er geen Z80 commando's meer volgen, het heeft dezelfde betekenis als het 'END' statement in BASIC.

Het programma vertaald naar machinetaal:

```
0
1 3EH,20H        Op adres A000H-A001H
2 32H,00H,90H    Op adres A002H-A004H
3 C9H            Op adres A005H
4
```

Uitleg:

regel 0: hier staat niets omdat het commando 'ORG' niets met de machinetaal zelf te maken heeft.

regel 1: 3EH betekent 'laad het A register met de byte direct volgend', in dit geval 20H.

regel 2: 32H zorgt er voor dat de inhoud van de accumulator wordt geladen in de geheugen locatie, bestaande uit twee bytes, volgend op 32H. Het lagere orde deel van de geheugen locatie staat direct achter 32H, het hogere orde deel staat achter het lagere orde deel.
In dit geval is de geheugen locatie 9000H. Het lagere orde deel bestaat uit 00H, het hogere orde deel uit 90H. 00H staat dus direct achter 32H en 90H staat achter 00H.

Algemene opmerking betreffende de behandeling van 16 bit adressen door de Z80:

De Z80 behandelt 16 bit adressen als 2 acht bit delen. Het lagere orde deel van een 16 bit adres staat altijd voor het hogere orde deel. De Z80 keert als het ware de twee acht bit delen om.

Voorbeeld:
Als u de Z80 een instruktie geeft om een 16 bit adres (b.v.3505H) of gegeven weg te zetten naar een bepaalde plaats A000H, dan plaatst de Z80 het lagere orde deel (05H) op adres A000H en het hogere orde deel (35H) op adres A001H.

regel 3: C9 staat voor return.

regel 4: Ook hier staat niets omdat het 'end' statement niets te maken heeft met de machinetaal.

Het meeste werk van het programma is nu achter de rug. We moeten nu alleen de gegevens in de computer brengen. Dit doen we onder BASIC m.b.v. het volgende programma.

```
10 DATA 3E,20,32,00,90,C9
20 CLEAR 200,&H8FFF
30 FOR X=&HA000 TO &HA005
40 READ A$:POKE X,VAL("&H"+A$)
50 NEXT
```

Na het 'runnen' van dit programma zitten alle gegevens in de computer om geheugen locatie 9000H met de waarde 20H te laden.

Uitleg:

regel 10: In deze regel staat de machinetaal

regel 20: Deze opdracht reserveert 200 bytes voor variabelen en reserveert geheugens 8FFFH en hoger voor machinetaal.

regel 40: De machinetaal wordt in het geheugen weggezet vanaf adres A000H.

Eindelijk kunnen we het machinetaal programma aanroepen. Ook dit gebeurt m.b.v. een BASIC commando.

Tik in: DEFUSR=&HA000:X=USR(0) en druk op de enter/return toets. Wat is er gebeurd?

'DEFUSR=&HA000' zorgt ervoor dat het PC register met de waarde A000H geladen wordt nadat het commando x=usr(0) gegeven is. De processor weet nu waar hij de eerste byte van van het programma kan halen, dat is dus op geheugen locatie A000H want daar staat de eerste byte (zie BASIC programma).
Na het commando X=USR(0) start het machinetaal programma.
We kunnen nu m.b.v. het BASIC 'peek' commando controleren of geheugen locatie 9000H inderdaad met de waarde 20H is geladen. Ga dit na met de instruktie PEEK(&H9000)

Het tweede programma:

Doel van het programma: We willen de geheugen locaties 9000H-90FFH met de waarde F0H laden.

Het assembler programma:

```
0 ORG A000H
1 LD A,FFH
2 LD HL,9000H
3 LOOP:LD (HL),F0H
```

```
4 INC HL
5 DEC A
6 JP NZ,LOOP
7 RET
8 END
```

Uitleg:

regel 3: 'LOOP:'

Dit is een label, u kunt dit vergelijken met een regel nummer in BASIC.
Het BASIC commando 'GOTO 10' heeft als effect dat het programma naar regel 10 springt. Onder de assembler kunt u ook commando's geven die hetzelfde effect hebben als het 'GOTO' statement in BASIC. I.p.v regel nummers kunt u labels aanbrengen.

LD (HL),FOH

Dit commando heeft als effect dat het getal FOH naar de geheugen plaats wordt geschreven waar HL naar verwijst.
Als HL de waarde AOOOH bevat, wordt geheugen locatie AOOOH met de waarde FOH geladen.

regel 4: INC HL

Heeft als effect dat de inhoud van registerpaar HL met 1 verhoogd wordt.

regel 5: DEC A

Heeft als effect dat de inhoud van register A met 1 verlaagd wordt.

regel 6: JP NZ,LOOP

Heeft als effect dat als het resultaat van de vorige berekening, in dit geval DEC A, ongelijk aan nul is (NZ=not zero), er naar het label LOOP wordt gesprongen. Is het resultaat van de vorige berekening wel nul, dan loopt het programma gewoon verder.
U kunt het 'JP NZ,LOOP' statement vergelijken met 'IF X<>0 THEN GOTO 10' bij BASIC, er wordt alleen naar regel 10 gesprongen als X ongelijk aan nul is. X is het resultaat van de vorige berekening.

De assembler taal vertaald naar machinetaal:

regel 1:	3E,FF	Op adres A000H-A001H
regel 2:	21,00,90	Op adres A002H-A004H
regel 3:	36,F0	Op adres A005H-A006H
regel 4:	23	Op adres A007H
regel 5:	3D	Op adres A008H
regel 6:	C2,05,A0	Op adres A009H-A00BH
regel 7:	C9	Op adres A00CH

Uitleg:

regel 1: 3EH EN FFH worden respectievelijk op adres A000H en A001H gezet.

regel 3: 36H wordt op adres A005H gezet, F0H op adres A006H

regel 6: C2H,05H,A0H betekent voor de Z80 spring naar adres A005 als het resultaat van de vorige berekening ongelijk aan nul is. Indien niet, ga dan verder.
Als het resultaat van de vorige berekening ongelijk aan nul is, dan wordt het PC register met de waarde A005H geladen zodat de volgende instruktie van dit adres wordt gehaald. Ook hier geldt weer dat eerst het lagere orde deel van het 16 bit adres direct achter C2H de dient te staan.

Het BASIC programma:

```
10 DATA 3E,FF,21,00,90,36,F0,23,3D
20 DATA C2,05,A0,C9
30 CLEAR 200,&H9000
40 FOR T=&HA000 TO &HA00C
50 READ A$:POKE T,VAL("&H"+A$):NEXT
60 DEFUSR2=&HA000
70 NEW
```

Het machinetaal programma kan nu worden aangeroepen van X=USR2(0).

Uitleg:

regel 10 en 20: Hier staan de gegevens voor de Z80

regel 30: reserveert 200 bytes voor variabelen en adres 9000H en hoger voor machinetaal.

regel 60: In BASIC kunnen tien verschillende start adressen voor machinetaal programma's gedefinieerd worden m.b.v. het 'DEFUSRX=ADRES' statement. Hierbij stelt X een nummer tussen de 0 en de 9 voor.

ADRES stelt het start adres van een machinetaal programma voor.
Er wordt als het ware een nummer aan een machinetaal programma toegekent.
DEFUSR2=&HA000 kent nummer 2 toe aan het machinetaal programma dat begint op adres A000H.

Deze programma's kunnen opgeroepen worden met het statement 'A=USRX(0)': Hierbij stelt A een BASIC variabele voor . X is het nummer van het machinetaal programma dat gestart dient te worden.
A=USR2(0) start het machinetaal programma waaraan nummer 2 is toegekent.

Verder met het volgende programma.

Doel: Het verplaatsen van een blok van 1000H bytes van de adressen 9000H-A000H naar A000H-B000H.

Om te laten zien dat er werkelijk een verplaatsing heeft plaats gevonden, dient u het onderstaand BASIC programma eerst te laten 'runnen'

```
5 DEFINT X
10 FOR X=&H9000 TO &HA000
20 POKE X,&HCC
30 NEXT
```

Dit BASIC programma zorgt ervoor dat de adressen 9000H t/m A000H met de waarde CCH geladen worden.

Het programma:

```
1 ORG   0B000H
2 LD HL,09000H
3 LD DE,0A000H
4 LD BC,01000H
5 LDIR
6 RET
7 END
```

Uitleg:

regel 1 t/m 4 moet u kunnen begrijpen.

regel 5:   LDIR

Dit is een instruktie die ervoor zorgt dat er een blok bytes verplaatst wordt. De grootte, de beginplaats waar het blok vandaan komt (source adres) en de beginplaats waar het blok naar toe gaat (destination adres) moeten van te voren in respectievelijk de BC, HL en DE registerparen geladen worden.

Dus: HL bevat de source adres

DE bevat het destination adres

BC bevat de grootte van het te verplaatsen blok.

Werking:

De inhoud van de geheugen locatie geadresseerd door HL, wordt in de geheugen locatie geadresseerd door DE geladen.

Daarna wordt bij DE en HL 1 opgeteld, van BC wordt 1 afgetrokken. Als BC ongelijk 0 is, dan wordt het PC register met 2 verminderd zodat het gehele proces zich herhaalt.

Naast 'LDIR' kent de Z80 nog een aantal blok instrukties te weten:

LDI

Deze instruktie verplaatst 1 byte van het geheugen adres geadresseerd door HL naar het geheugen adres geadresseerd door DE. Daarna wordt BC met 1 verminderd, HL en DE worden met 1 verhoogd.

LDD

Deze instruktie verplaatst 1 byte van het geheugen adres dat in HL staat naar het geheugen adres dat in DE staat. Daarna worden BC, HL en DE met 1 verminderd.

LDDR

Deze instruktie verplaatst ook een blok bytes. De lengte van dit blok dient in BC geladen worden, de hoogste plaats waar het blok vandaan komt dient in HL te staan en de hoogste plaats waar het blok naar toe moet in DE.
Nadat er een byte verplaatst is, worden DE, HL en BC met 1 verminderd.
Het verschil met 'LDIR' is dat de verplaatsing bij 'LDDR' begint op het hoogste adres van het blok i.p.v. het laagste zoals bij 'LDIR' het geval is, 'LDDR' werkt in het geheugen naar beneden terwijl

'LDIR' in het geheugen omhoog werkt.

Het assembler programma vertaalt naar machinetaal en BASIC.

N.B. Bij alle programma's die nu nog volgen wordt de machinetaal niet apart meer vermeldt. De machinetaal is te vinden in de BASIC regels die met 'DATA' beginnen.

```
10 DATA 21,00,90,11,00,A0,01,00
15 DATA 10,ED,B0,C9
20 CLEAR 200,&H9000
30 FOR T=&HB000 TO &HB00B
40 READ A$:POKE T,VAL("&H"+A$):NEXT
50 DEFUSR=&HB000
```

Nadat dit programma 'gerund' heeft, zitten alle gegevens voor het programma in het geheugen van de computer.

Na het ingeven van 'x=usr(0)' wordt het programma uitgevoerd.

Misschien gelooft u niet dat een blok van 1000H bytes zo snel verplaatst kan worden, daarom gaan we nu onder BASIC bekijken of dit inderdaad gebeurd is.
Voor we het machinetaal programma startten hebben we eerst (onder BASIC) geheugen locaties 9000H-A000H met de waarde CCH geladen. Als alles goed is verlopen, moeten geheugen locaties A000H-B000H nu ook deze waarde bevatten.

Tik in:

```
10 FOR X=&HA000 TO&HB000:PRINT HEX$(PEEK(X));:NEXT
```

Als het goed is, verschijnen er allemaal C's in beeld.

Machinetaal wegschrijven

Nu we wat verder zijn gevorderd met onze programma's zou het wel eens leuk zijn als we er enkele op tape/disk zouden kunnen bewaren. Uw BASIC programma's kunt u het best bewaren met het commando CSAVE, waarna u ze met CLOAD weer kunt inlezen. Het SAVE commando schrijft uw BASIC programma weg in het z.g. ASCII formaat. (spreek uit askie) Hierbij wordt als het ware uw hele programma als een tekst beschouwd. Zoals u weet heeft uw MSX voor elk teken een code (ASCII-code). Deze codes worden dan ook bij een SAVE instruktie opgeslagen i.p.v. de BASIC tokens (CSAVE). Dit kan bij het overbrengen van uw programma via een modem naar een niet-MSX computer zijn diensten bewijzen. Ook bij het 'mergen' van programma's bewijst dit SAVE commando zijn diensten.

Bij het wegschrijven van machinetaalprogramma's hebben we echter veel meer aan het commando BSAVE. Dit commando stelt u in staat om gedeelten van het (video)geheugen weg te schrijven. We zullen hiervan een voorbeeld geven.

Voorbeeld:
Stel uw machinetaal programma begint op C000H en eindigt op C500H em het startadres heeft u bepaald op C010H.
Dan dient u dit weg te schrijven met:

```
BSAVE"NAAM",&HCOOOH,&HC5OO,&HCO10
                  !         !        !
begin adres ---!         !        !
eind  adres -----------!        !
start adres ------------------!
```

U kunt uw programma dan weer laden door in te tikken:

BLOAD"NAAM"

Door achter dit BLOAD statement een ',R' te plaatsen wordt uw programma automatisch 'gerund'.
Voorbeeld:

BLOAD"NAAM",R

Wilt u uw programma op een ander adres in het geheugen laden dan het normale beginadres, dan kan dat ook d.m.v. het BLOAD commando. Stel dat u een programma niet op het beginadres COOOH maar op het adres DOOOH wilt zetten, dan doet u dit als volgt:

BLOAD"NAAM",+&H1OOO

Hetzelfde geldt als u het programma lager in het geheugen wilt zetten dan het begin adres. Nu dient u echter i.p.v. de '+' een '-' teken te gebruiken.

Om een deel van het Video RAM weg te kunnen schrijven moeten we i.p.v. het startadres een 's' noteren (s=screen). Let u er echter wel op dat het Video RAM maar

16K groot is. Op deze manier kunt u bijvoorbeeld een grafisch scherm wegschrijven.
Voorbeeld:

BSAVE"scherm",&H0000,&H3FFF,S

Om een scherm te laden dient u achter het commando BLOAD een 'S', i.p.v. een 'R' te plaatsen.

Het d.m.v. bovenstaand commando wegschrijven van het Video RAM is alleen bruikbaar als uw MSX computer is voorzien van een diskdrive. Voor een cassetterecorder moeten we een andere oplossing zien te zoeken.

Hieronder vindt u twee routines die er voor zorgen dat ook de cassette gebruikers de mogelijkheid hebben om het grafische scherm weg te schrijven en weer in te laden van tape.

Het wegschrijven:

```
1 REM SCHERM WEGSCHRIJVEN
2 :
5 CLEAR 200,&HCFFF
10 FOR I=&HD000 TO &HD000+24
20 READ A$:A=VAL("&H"+A$)
30 POKE I,A:NEXT
40 DEFUSR=&HD000
50 DATA 21,00,00,11,40,9C,01,00,18,CD,59,0
0,21,00,20,11,40,B4,01,00,18,CD,59,00,C9
60 NEW
```

Het wegschrijven van een scherm naar cas-

settte gebeurt nu op de volgende manier:

```
10 SCREEN2
20 CIRCLE(128,96),60,15
30 X=USR(0):BSAVE"SCHERM",&H9C40,&HCC40
40 END
```

Het laden:

```
1 REM       SCHERM LADEN
2 :
5 CLEAR 200,&HCFFF
10 FOR I=&HD000+25 TO &HD000+49
20 READ A$:A=VAL("&H"+A$)
30 POKE I,A:NEXT
40 DEFUSR1=&HD000+25
50 DATA 21,40,9C,11,00,00,01,00,18,CD,5C,0
0,21,40,B4,11,00,20,01,00,18,CD,5C,00,C9
60 NEW
```

Het laden gaat nu als volgt: 'Run' het bovenstaand programma en voer dan de instruktie 'X=USR1(2)' uit zoals gedemonstreerd in onderstaand programma;

```
10 SCREEN2
20 BLOAD"SCHERM":X=USR1(2)
30 GOTO 30
```

Binair optellen (in machinetaal)

Dit programma telt twee acht bits getallen op, het resultaat is ook acht bits groot. Het eerste getal staat op adres A000H, het tweede op A001H terwijl de som op adres A002H komt te staan.

Het assembler programma:

```
1 ORG 9000H
2 LD HL,0A000H
3 LD A,(0A001H)
4 ADD A,(HL)
5 LD (0A002H),A
6 RET
7 END
```

Uitleg:

Regel 3: LD A,(0A000H)

Deze instruktie laadt register A met de byte die op geheugenplaats 0A000H staat (let op de haakjes), A wordt geladen met de byte geadresseerd door 0A000H.

regel 4: ADD A,(HL)

Deze instruktie telt (ADD is optellen) de byte geadresseerd door HL op bij de inhoud van het A register. Het resultaat wordt in het A register gezet.
HL bevat in dit geval de waarde A000H, op deze plaats staat de ene term van de op-

telling. De accumulator bevat de andere term.

regel 5: LD (0A002H),A

Deze instruktie zet de inhoud van het A register weg naar geheugen locatie 0A002H. Zoals afgesproken komt het resultaat op adres A002H te staan.

Het BASIC programma:

```
10 DATA 21,00,A0,3A,01,A0,86,32,02
20 DATA A0,C9
25 CLEAR 200,&H8FFF
30 FOR X=&H9000 TO &H900A
40 READ A$:POKE X,VAL ("&H"+A$):NEXT
50 DEFUSR=&H9000
```

Nadat u dit programma heeft gerund, kunt u met het statement 'x=usr(0)' het programma oproepen, echter u wilt natuurlijk wel zien dat de optelling inderdaad plaats heeft gevonden. Dit kunt u doen door adres A000H met bijvoorbeeld de waarde 33 te laden en adres A001H met 66. Na het aanroepen van het machinetaal programma staat het resultaat op adres A002H.

Het vorige programma was beperkt omdat slechts twee getallen, niet groter dan acht bits, bij elkaar opgeteld konden worden. Met het volgende programma is het mogelijk om twee 16 bits getallen op te tellen.

Het volgende programma realiseert een 16 bits optelling. Daartoe worden de twee 16 bits getallen opgesplitst in twee acht

bits getallen, in een lager- en hoger orde deel.
Het eerste getal staat op de adressen A000H en A001H. Het lagere orde deel staat op adres A000H, het hogere op A001H.
Het tweede getal staat op de adressen A002H en A003H.
De uitkomst komt op de adressen A004H en A005H te staan.

```
1  ORG 9000H
2  LD A,(0A000H)  ;LAAD HET LAGERE ORDE DEEL
                  ;VAN GETAL1 IN A
3  LD HL,0A002H   ;LAAD HET ADRES VAN  HET
                  ;LAGERE ORDE DEEL VAN
                  ;GETAL2 IN HL
4  ADD A,(HL)     ;TEL DE LAGERE ORDE DELEN
                  ;BIJ ELKAAR OP
5  LD (0A004H),A  ;ZET HET RESULTAAT WEG  OP
                  ;ADRES A004H
6  LD A,(0A001H)  ;LAAD HET HOGERE ORDE DEEL
                  ;VAN GETAL1 IN A
7  INC HL         ;VERWIJS HL NAAR HET ADRES
                  ;VAN HET HOGERE ORDE DEEL
                  ;VAN GETAL2
8  ADC A,(HL)     ;TEL DE HOGERE ORDE DELEN
                  ;MET EVENTUELE CARRY OP

9  LD (0A005H),A  ;ZET  HET  RESULTAAT  OP
                  ;ADRES A005H
10 RET            ;TERUG NAAR BASIC
11 END
```

Het bovenstaande programma bevat opmerkingen. Alle tekens achter de ';' worden door de assembler genegeerd. (vergelijk met 'REM' in BASIC).
Bij de meeste assembler programma's vindt

u opmerkingen. Dit is gedaan om het geheel voor u zelf en anderen leesbaar te houden.

Uitleg:

In regel 4 worden de lagere orde delen bij elkaar opgeteld. Als de som een getal is dat de 8 bits overschrijdt, dan ontstaat er een carry. Deze carry wordt door de Z80 in een speciaal register opgeslagen, het F of FLAG register. Meer hierover later in dit boek.

regel 8: ADC A,(HL)

Deze instruktie telt de inhoud van het A register met carry op bij de byte geadresseerd door het HL register paar.
In dit programma worden de twee hogere orde delen van getal-1 en getal-2 bij elkaar opgeteld door deze instruktie. Als er een carry onstaan is bij de lagere orde optelling, dan wordt deze verwerkt bij de hogere orde optelling.

Het BASIC poke programma:

```
10 DATA 3A,00,A0,21,02,A0,86,32,04
20 DATA A0,3A,01,A0,23,8E,32,05,A0
30 DATA C9
35 CLEAR 200,&H8FFF
40 FOR X=&H9000 TO &H9012
50 READ A$:POKE X,VAL("&H"+A$)
60 NEXT:DEFUSR9=&H9000
```

Het machinetaal programma kan nu opgeroepen worden met x=usr9(0).
Eerst moet u er natuurlijk voor zorgen dat

u de twee op te tellen getallen op de juiste plaats in het geheugen zet. Hieronder volgt een voorbeeld.

Stel we willen de getallen 700 en 333 bij elkaar optellen. Als eerste moeten we deze decimale getallen omzetten naar het hexadecimale talstelsel omdat we ieder getal moeten opsplitsen in twee delen.

700 decimaal levert 02BCH
333 decimaal levert 014DH

Als we het getal 02BCH opsplitsen in een hoger en lager orde deel, dan ontstaan respectievelijk de getallen BCH en 02H. Splitsen we 014D op dan onstaan de getallen 4DH en 01H.

Deze delen moeten we nu in het geheugen zetten. Dit doen we d.m.v. de BASIC 'poke' instruktie:

Tik in:

```
POKE &HA000,&HBC + (ENTER)
POKE &HA001,&H02 + (ENTER)
POKE &HA002,&H4D + (ENTER)
POKE &HA003,&H01 + (ENTER)
```

Nu beide getallen op de juiste plaatsen in het geheugen staan, kunnen we het machinetaal programma oproepen. Als dit gebeurd is, dan hoeven we alleen het resultaat van de optelling uit het geheugen te halen:

Tik in:

```
PRINT PEEK (&HA004) + (ENTER)
PRINT PEEK (&HA005) + (ENTER)
```

Zet nu het resultaat van de eerste peek instruktie (09) achter het resultaat van de tweede peek instruktie (04). Het getal dat nu onstaan is (0409H), is de uitkomst van de optelling.
Wilt u de uitkomst decimaal bekijken, dan moet u onder BASIC ingeven: PRINT &H0409.

Voordat we verder gaan met binair aftrekken en vermenigvuldigen, behandelen we eerst het VLAG register.

Het F of vlag register

Dit register is al eerder ter sprake gekomen. Het is een zeer belangrijk register omdat het wordt gebruikt bij het testen van voorwaarden. De systematische opbouw is weergegeven in onderstaand figuur.

```
+----+----+----+----+----+----+----+----+
+ S  + Z  +    + H  +    + P/V+ N  + C  +
+----+----+----+----+----+----+----+----+
  7    6    5    4    3    2    1    0
```

C   - CARRY OVERDRACHT
N   - AFTREKKEN
P/V - PARITEIT/OVERLOOP
H   - HALVE OVERDRACHT
Z   - ZERO (NUL)
S   - SIGN

Uit de figuur blijkt dat we te maken hebben met een acht bits register. Elke bit vertegenwoordigt een bepaalde vlag.

In dit boek wordt alleen gebruik gemaakt van bit 0 en bit 6, daarom worden alleen deze twee behandeld.

BIT 0

Dit is het CARRY (C) bit.

Dit bit wordt beinvloed door alle rekenkundige instrukties. Afhankelijk van de uitkomst wordt het bit op een 0 of 1

gezet.

Het bit wordt op 0 gezet door alle logische functies (AND, OR, XOR).

Verder dient het als negende bit bij schuif en roteer functies (nog niet behandeld).

BIT 6

Dit bit, ook wel de Zero (Z) vlag genoemd, geeft aan of de waarde van een berekende of verplaatste byte 0 is.
Is het resultaat van een bepaalde bewerking of data verplaatsing gelijk aan 0, dan wordt de zero vlag 1 gemaakt. In het andere geval wordt Z een 0.

Verder wordt het bit geset als bij een BIT instruktie (niet behandeld) de opgegeven bit gelijk is aan 0.

BELANGRIJK:

Instrukties die het ZERO bit niet beinvloeden zijn:

ADD DD,SS

(DD en SS staan voor register paren b.v. 'add HL,BC', tel de inhoud van HL en BC bij elkaar op en plaats het resultaat in HL).

INC DD

(vermeerder register paar DD met 1).
Bijvoorbeeld: INC HL, na deze instruktie

is de inhoud van HL met 1 verhoogd.

DEC DD

(verminder register paar DD met 1).
B.v. INC DE, na deze instruktie is de inhoud van DE met 1 verlaagd.

Binair aftrekken

Het volgende programma realiseert een 16 bits aftrekking. Getal-2 wordt van getal-1 afgetrokken. Getal-1 dient op de adressen A000H en A001H te staan, getal-2 op A002H en A003H. De uitkomst komt op adressen A004H en A005H te staan. Getal-1 moet groter dan getal-2 zijn.
De Z80 kent twee soorten optel instrukties namelijk 'ADD' en 'ADDC'. Deze instrukties zijn bij het optellen behandeld. Er is echter maar 1 aftrek instruktie: 'SBC', trek af met carry.

SBC HL,DE

heeft als effect dat register paar DE met carry (indien aanwezig) van HL afgetrokken wordt. Het resultaat wordt in HL gezet.

Het programma.

```
1 ORG 9000H
2 LD HL,(0A000H)  ; GETAL1 IN HL
3 LD DE,(0A002H)  ; GETAL2 IN DE
4 XOR A           ; MAAK CARRY BIT 0
5 SBC HL,DE       ; TREK GETAL2 VAN GETAL1 AF
6 LD (0A004H),HL  ; ZET RESULTAAT WEG
7 RET             ; TERUG NAAR BASIC
9 END
```

Uitleg:

regel 4: 'XOR A' zorgt ervoor dat het carry bit nul wordt gemaakt, doen we dit

niet, dan krijgen we een verkeerd resultaat als het carry bit een 1 is omdat er afgetrokken wordt met carry.

Het BASIC programma:

```
10 DATA 2A,00,A0,ED,5B,02,A0,AF
20 DATA ED,52,22,04,A0,C9
30 CLEAR 200,&H8FFF
40 T=&H9000
50 FOR X=T TO T+13
60 READ A$
70 POKE X,VAL("&H"+A$):NEXT
80 DEFUSR=&H9000
```

Op adressen A000H en A001H dient getal-1 te staan (in omgekeerde volgorde), op A002H en A003H getal-2. Op A004H en A005H komt het antwoord.

Binair vermenigvuldigen

(de schuifinstrukties)

Binair vermenigvuldigen is moeilijker dan aftrekken en optellen omdat de Z80 geen vermenigvuldig instrukties kent. We moeten dus zelf een routine schrijven. In onderstaand programma is zo'n routine te vinden. Het realiseert een 8 bits vermenigvuldiging.
Het programma is bedoeld om met de schuifinstrukties te leren omgaan. In het programma wordt getal A met getal B vermenigvuldigd, de uitkomst is getal C. Getal A dient op A000H te staan, getal B op A001H terwijl de uitkomst op A002H en A003H komt te staan (een 16 bits getal).

OPMERKING: In principe is het mogelijk om het programma korter te maken, echter daardoor wordt het geheel moeilijker te begrijpen.

Een vermenigvuldig programma;

```
ORG 9000H
LD BC,(0A000H);LAAD GETAL B IN REG. C
LD B,08H      ;LAAD REG B MET 08H
LD DE,(0A001H);LAAD GETAL A IN REG. E
LD D,00H      ;MAAK REG D GELIJK AAN 0
LD HL,00H     ;MAAK GETAL C 0
LABELA:SRL C  ;SCHUIF  BIT IN CARRY
JR NC,LABELB  ;INDIEN GEEN  CARRY GA
              ;NAAR LABELB
ADD HL,DE     ;INDIEN CARRY 1 WAS, TEL
```

```
                  ;DAN GETAL A OP BIJ HL
10 LABELB:SLA E  ;SCHUIF GETAL A NAAR LINKS
11 RL D          ;RED UITGESCHOVEN BIT IN D
12 DEC B         ;NOG B V.D. 8 BITS TE GAAN
13 JR  NZ,LABELA;INDIEN B ONGELIJK O  DAN
                 ;NAAR LABELA
14 LD (0A002H),HL;SCHRIJF UITKOMST WEG
15 RET            ;TERUG NAAR BASIC
16 END
```

In dit programma komen een aantal nieuwe instrukties voor. We zullen deze eerst behandelen.

SRL r

r staat voor een register. In dit programma gebruiken we SRL C (regel 7). Dit is een instruktie die voor een verschuiving zorgt. De instruktie heeft als effect dat bit 0 van het C register in de CARRY vlag terecht komt, bit 1 komt in bit 0, bit 2 in bit 1 enz., tenslotte wordt er een 0 in bit 7 geschoven. Grafisch voorgesteld levert dit het volgende plaatje. Stel het carry bit en register C bevatten de volgende waarde:

```
 bit nr. 7 6 5 4 3 2 1 0   Carry
         ----------------- ---
reg C    =1=0=1=1=0=1=0=1= =0=
         ----------------- ---
```

Dan is de waarde van het carry bit en register C na de instruktie als volgt:

```
 bit nr. 7 6 5 4 3 2 1 0   Carry
         ----------------- ---
reg C    =0=1=0=1=1=0=1=0= =1=
         ----------------- ---
```

SLA r

r staat voor een register.
Ook deze instruktie zorgt voor een bit verschuiving. In ons geval SLA E. Bit nr. 7 wordt in de carry geschoven, bit 6 in bit 7, bit 5 in bit 6 enz., bit 0 uiteindelijk wordt met een 0 geladen. We komen zo tot het volgende plaatje.

Stel register E en het Carry bit bevattten onderstaande waarde.

```
  Carry. bit nr. 7 6 5 4 3 2 1 0
   ---           ---------------
   =0=           =0=1=0=1=1=0=1=0=  reg E
   ---           ---------------
```

Na de instruktie SLA E is dat veranderd in:

```
  Carry. bit nr. 7 6 5 4 3 2 1 0
   ---           ---------------
   =0=           =1=0=1=1=0=1=0=0=  reg E
   ---           ---------------
```

RL r

Deze instruktie roteert register r (in dit geval register D) en de carry vlag naar links. De inhoud van de carry vlag komt in bit 0, bit 0 komt in bit 1, bit 1 in bit 2 enz., bit 7 uiteindelijk wordt in de carry vlag geschoven. We komen zo tot de volgende voorstelling.
Stel register D en het carry bit bevatten de volgende waarde.

```
Carry. bit nr. 7 6 5 4 3 2 1 0
 ---            -----------------
 =1=            =0=0=1=1=1=1=0=0=  reg D
 ---            -----------------
```

Na de instruktie RL D verkrijgen we het volgende resultaat.

```
Carry. bit nr. 7 6 5 4 3 2 1 0
 ---            -----------------
 =0=            =0=1=1=1=1=0=0=1=  reg D
 ---            -----------------
```

Nu we deze instrukties behandeld hebben, gaan we verder met de uitleg van het programma.

Regels 2-6: Initialisatie, alle registers worden met nodige waarden geladen.

Reg. C wordt via 'LD BC,(0A000H)' geladen omdat de instruktie 'LD C,(0A000H)' niet bestaat. Dit geldt evenzo voor het laden van register E.

Reg. B wordt met 8H geladen omdat een byte 8 bits heeft, we moeten de 8 bits van getal B testen op een nul of een 1.

Reg. paar HL wordt 0 gemaakt omdat hierin het resultaat komt. Dit moet voor de vermenigvuldiging 0 zijn.

Regel 7: SRL C: Deze instruktie schuift bit 0 van getal B in de carry. De inhoud van bit 1 wordt in bit 0 geschoven, bit 2 komt in bit 1 enz. In bit 7 komt een 0.

Regel 8: JR NC,LABELB: Deze instruktie springt naar LABELB als het carry bit ongelijk aan 0 is, dus als bit 0 van register C een 0 was.

Regel 9: ADD HL,DE: Telt HL en DE bij elkaar op, het resultaat komt in HL te staan. Deze optelling moet alleen maar plaatsvinden als het carry bit een 1 was.

Regel 10: SLA E: Deze instructie schuift de inhoud van getal A naar links. Bit 7 van getal A komt in de carry, bit 0 wordt een 0.

Regel 11: Het zevende bit van getal A dat door de vorige instruktie in de carry is gekomen, wordt nu in register D geschoven.

Regels 12 en 13: Voert het geheel nog een keer uit als niet alle 8 bits van getal B aan de beurt zijn geweest.

Regel 14: Zet het uiteindelijke resultaat op geheugenplaats A002H en A003H.

Een getallen voorbeeld:
Stel we willen het getal 00001111 (getal A) met het getal 00000111 (getal B) vermenigvuldigen.

```
        00001111        A
        00000111 *      B
       ------------
        00001111       (d) HL=15
       000011110       (e) HL=45
      0000111100       (f) HL=105
     00000000000           HL=105
    000000000000           HL=105
   0000000000000           HL=105
  00000000000000           HL=105
 000000000000000 +         HL=105
--------------------
 000000001101001
```

Getal (d) is gelijk aan getal A. Bit 0 van getal B is een 1 dus schrijven we getal A op oftewel we laden het in register paar HL . Getal (e) is getal A echter bit 0 is geen 1 maar een 0. We hebben als het ware een nul aangehaald (net zoals bij gewoon vermenigvuldigen), de instrukties SLA E ('SLA getal A') en RL D bewerkstelligen dat. Getal (e) (Dit getal zit in register paar DE) tellen we bij HL op omdat bit 1 van getal B een 1 is.
Getal (f) is getal A met twee nullen extra. Ook dit getal tellen we op bij HL.
Daar de overige bits van getal B gelijk aan 0 zijn, verandert er niets meer aan de inhoud van register paar HL.

Het BASIC poke programma

```
10 DATA ED,4B,00,A0,06,08,ED,5B
20 DATA 01,A0,16,00,21,00,00,CB
```

```
30 DATA 39,30,01,19,CB,23,CB,12
35 DATA 05,20,F4,22,02,A0,C9,**
40 CLEAR 200,&H87FF
60 T=&H9000
70 READ A$:IF A$="**" THEN 80 ELSE PO
KE T,VAL("&H"+A$):T=T+1:GOTO 70
80 DEFUSR=&H9000
```

Nadat dit programma gerund is, kan het machinetaal programma met het x=usr(0) statement aangeroepen worden. Dit heeft natuurlijk alleen zin als geheugen locaties A000H en A001H met een bekende waarde geladen zijn.

B.v.

```
POKE &HA000,30
POKE &HA001,2
X=USR(0)
```

Na deze instrukties bevatten A002H en A003H het produkt. Het lagere orde deel staat op A003H, het hogere orde deel op A002H.

Sprong instrukties (JR/JP)

Er zijn twee verschillende sprong instrukties die onder voorwaarde uitgevoerd worden: de absolute en relatieve sprong instrukties.

A: de absolute

JP CC,PQ

Deze instruktie zijn we al tegengekomen. PQ staat voor een geheugen adres. CC (de voorwaarde) kan zijn:

NZ	niet nul
Z	nul
NC	geen carry
C	carry

De instruktie heeft als effect dat als er aan de voorwaarde (CC) voldaan wordt, het PC register met de waarde PQ wordt geladen. Het programma gaat dan verder vanaf adres PQ, van adres PQ wordt de volgende instruktie gehaald.

Voorbeelden:

```
JP  C,AOOOH ; SPRING NAAR ADRES AOOOH ALS
            ; HET CARRY BIT VAN HET VLAG
            ; REGISTER EEN  1 IS.

JP NZ,BOOOH ; SPRING NAAR ADRES BOOOH ALS
            ; HET ZERO BIT VAN HET VLAG
```

```
        ; REGISTER ONGELIJK 0 IS.
```

B: de relatieve

JR CC,E

Deze instruktie springt E plaatsen verder of terug als aan de voorwaarde CC voldaan wordt. Het is een relatieve sprong, de waarde van het PC register na deze instruktie hangt af van de waarde voor de instruktie.

E is een getal dat in het twee complement genoteerd wordt.

CC kan zijn:

NZ, Z, NC, C

Als aan de gespecificeerde voorwaarde (CC) voldaan wordt, dan wordt het opgegeven getal E opgeteld bij de PC register. Daar de instruktie zelf twee bytes in beslag neemt, is de waarde van de PC register, nadat de instruktie heeft plaats gevonden, 2 hoger geworden. Dit laatste is alleen van belang voor degene die niet over een assembler beschikken.

Voorbeelden:

```
JR NC,LOOP ;SPRINGT NAAR HET LABEL 'LOOP'
           ;ALS HET CARRY BIT 0 IS. HET
           ;LABEL  'LOOP' MAG  NIET MEER
           ;DAN  129  PLAATSEN  VERDER
           ;LIGGEN. BOVENDIEN MAG 'LOOP'
           ;NIET MEER DAN 126 PLAATSEN
```

```
              ;TERUG IN HET GEHEUGEN LIGGEN.

JR  NZ,FBH    ;SPRING VIJF PLAATSEN TERUG
              ;ALS DE ZERO VLAG ONGELIJK
              ;AAN 0 IS (FBH IS -5 VOL-
              ;GENS HET 2 COMPLEMENT).
```

DJNZ E

Deze instruktie vermindert de inhoud van register B met 1, als het resultaat ongelijk aan 0 is, dan wordt het getal E (genoteerd in het 2 complement) bij het PC register opgeteld.
Met deze instruktie kunt u op een korte manier een 'loop' maken.

Voorbeeld:

```
DJNZ LABEL ;ALS REGISTER B ONGELIJK AAN  0
           ;IS NADAT ER 1 IS AFGETROKKEN,
           ;DAN WORDT ER NAAR HET LABEL
           ;'LOOP' GESPRONGEN.'LOOP' MAG
           ;NIET MEER  DAN  126 PLAATSEN
           ;TERUG OF 129 PLAATSEN VERDER
           ;LIGGEN.

DJNZ 04H
```

Naast voorwaardelijke sprong instrukties, kent de Z80 ook onvoorwaardelijke sprong instrukties. Hieronder volgen de belangrijksten;

JP PQ

Spring naar adres PQ.

Deze instruktie laadt de Programma Teller (PC register) met de waarde PQ. De volgende instrukties worden dus vanaf adres PQ gehaald.

V.B.

JP OAOFFH ;SPRING NAAR ADRES AOFFH

JR E

Spring E plaatsen relatief.
E is een getal genoteerd in het 2 complement.

Het PC register wordt met de de waarde E gesommeerd (-128<E<127), het resultaat wordt in het PC register gezet. Er kunnen dus maximaal 128 plaatsen worden terug geprongen of 127 plaatsen verder.

Voorbeeld:

JR FAH ;SPRING 6 PLAATSEN TERUG.

JP (HL)

Spring naar HL;

De inhoud van het register paar HL wordt in het PC register geladen. De volgende instruktie wordt van dit adres gehaald.

Het compare commando(CP)

Met het compare commando is het mogelijk om de inhoud van de accumulator te vergelijken met een andere byte. Het gespecificeerde byte wordt nl. van het A-register afgetrokken en het resultaat wordt dan genegeerd.

- Als de andere byte groter is dan de inhoud van de accumulator, dan wordt de Carry-vlag geset (wordt 1).

- Als de andere byte gelijk is aan die in de accumulator, dan wordt de nul-vlag geset.

- Als de andere byte kleiner of gelijk is aan de inhoud van de accumulator, dan wordt de carry vlag 'gereset' (wordt 0).

Hieronder een kort overzicht.
(alleen voor getallen tussen de 0 en 255!)

	Accumulator inhoud
Carry vlag = 0	>=
Nul vlag = 0	=
Carry vlag = 1	<
Nul vlag = 1	<>
Carry = 0 & Nul = 1	>
Carry = 1 & Nul = 0	=<

Voorbeeld:

```
     Stel Accumulator bevat FEH
          Register B  bevat 8FH
```

CP B

Resultaat:

```
     11111110
     10001111
     -------- -
     01101111
```

```
Geen carry bit ==>> carry vlag (C) = 0
Bit 7 = 0      ==>> Teken vlag (S) = 0
<> 0           ==>> Nul vlag   (Z) = 0
```

De carry vlag is gelijk aan 0, dus is de inhoud van de accumulator groter dan de inhoud van het B-register en dat klopt.

De LD (LOAD) instrukties

Aanvankelijk is het tijdens het programmeren in machinetaal vrij moeilijk om een laad instruktie te vinden die de Z80 herkent. Daarom volgt er hieronder een overzicht van de meest belangrijke laad instrukties. M.b.v een voorbeeld worden ze uitgelegd.

LD DD,(NN)

DD staat voor een register paar, NN staat voor een geheugen locatie.

Bijvoorbeeld: LD BC,(08000H)
Dit heeft als effect dat register C geladen wordt met de byte op adres 8000H, register B wordt geladen met de byte op adres 8001H.

LD DD,NN

DD staat voor een register paar, NN staat voor een gegeven.

Bijvoorbeeld: LD HL,5000H
Na deze instruktie is register paar HL met de waarde 5000H geladen.

LD R,N

R staat voor een register, N staat voor een gegeven.

Bijvoorbeeld: LD D,FFH
Register D wordt met de waarde FFH geladen.

LD R,R'

R voor een register, R' ook.

Bijvoorbeeld: LD A,H
Register A wordt met de inhoud van register H geladen.

LD (BC),A

De geheugen locatie geadresseerd door BC wordt met de inhoud van A geladen.

Bijvoorbeeld:
Stel BC is geladen met 8000H en A met CCH.
Na de instruktie bevat geheugen locatie 8000H de waarde CCH.

LD (DE),A

Zie vorige instruktie

LD (HL),N

N staat voor een gegeven.

Bijvoorbeeld: LD (HL),EEH
Na deze instruktie bevat geheugen locatie HL de waarde EEH

LD (HL),R

R staat voor een register.

Bijvoorbeeld: LD (HL),L
Na deze instruktie bevat geheugen locatie HL de inhoud van register L.

LD A,(NN)

NN staat voor een geheugen locatie.

Bijvoorbeeld: LD A,(5000H)
Dit heeft als effect dat A geladen wordt met de inhoud van geheugenlocatie 5000H.

LD (NN),A

NN staat voor een geheugen locatie.

Bijvoorbeeld: LD (9500H),A
Na deze instruktie bevat geheugenlocatie 9500H de waarde die in A staat.

LD (NN),DD

NN staat voor een geheugen locatie, DD staat voor een register paar.

Bijvoorbeeld: LD (9000H),BC
Dit heeft als effect dat geheugen locatie 9000H met de inhoud van register C wordt geladen, geheugen locatie 9001H wordt met de inhoud van register B geladen.

LD A,(BC)

Bijvoorbeeld:
Stel dat BC de waarde A000H bevat en geheugen locatie A000H bevat CCH, dan bevat register A na deze instruktie de waarde CCH.

LD A,(DE)

Zie vorige instruktie.

LD HL,(NN)

NN staat voor een geheugen locatie.

Bijvoorbeeld: LD HL,(8000H)
Na deze instruktie bevat register L de inhoud van geheugen locatie 8000H, register H bevat de inhoud van geheugen locatie 8001H.

LD R,(HL)

R staat voor een register.

Bijvoorbeeld: LD A,(HL)
Na deze instruktie bevat register A de inhoud van geheugen locatie geadresseerd door HL.

Op de volgende bladzijde vindt u alle bovenstaande LOAD instrukties in een twee verschillende tabellen gezet. Tabel 1 bevat de 8 bit instrukties, tabel 2 bevat de 16 bit instrukties.

TABEL 1

GG

FF	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(NN)	N
A	X	X	X	X	X	X	X	X	X	X	X	X
B	X	X	X	X	X	X	X	X				X
C	X	X	X	X	X	X	X	X				X
D	X	X	X	X	X	X	X	X				X
E	X	X	X	X	X	X	X	X				X
H	X	X	X	X	X	X	X	X				X
L	X	X	X	X	X	X	X	X				X
(HL)	X	X	X	X	X	X	X					
(BC)	X											
(DE)	X											
(NN)	X											X

NN staat voor een 16 bits getal bijvoorbeeld 0085H (=85H).
N staat voor een 8 bits getal bijvoorbeeld 40H.

TABEL 2

GG

FF	BC	DE	HL	SP	NN	(NN)
BC					X	X
DE					X	X
HL					X	X
SP			X		X	X
(NN)	X	X	X	X		

NN staat voor een 16 bits getal.

Hoe zijn de tabellen opgebouwd?

De tabellen moeten volgens het principe 'LD GG,FF' gelezen worden. Hierbij stelt GG een register (paar) uit de kolom voor en FF een register (paar) uit de rij.

```
Voorbeelden: LD A,N (laad A met data N)
             LD (HL),L
             LD (BC),A
             LD (NN),N
             LD BC,NN
             LD HL,(NN)
```

Uit de tabel 1 valt direct af te lezen dat de instruktie 'LD H,(BC)' niet bestaat.

Communicatie met randapparatuur

Een interrupt is een verzoek van een randapparaat aan de Z80 om bediend te worden.
De video processor geeft vijftig keer z'n interrupt. Als dit gebeurt, dan maakt de Z80 eerst de instruktie af waarmee hij bezig is, vervolgens zet hij de programma counter (PC register) op de stapel en als laatste 'vraagt hij' wat de video processor te vertellen heeft.

De video processor wil dat de programma teller met 038H geladen wordt, dit realiseert hij door een instruktie aan de Z80 door te geven die daar voor zorgt.
Als het PC register eenmaal met 038H geladen is, dan is de video processor 'uitgepraat', de Z80 gaat verder vanaf 038H. Dit houdt ondermeer in dat er wordt gekeken of er een toets op het toetsenbord ingedrukt wordt, het toetsenbord wordt 'afgescand'. Vervolgens wordt het eigenlijke programma weer vervolgd, net zolang tot er weer een interrupt signaal verschijnt.
Als u machinetaal programma's schrijft, dan is het meestal beter om de Z80 in een mode te dwingen zodat hij interrups negeert. Daartoe bestaat er een speciale instruktie.

DI

Deze instruktie, DI staat voor Disable Interrupt. De interrupt mode wordt uitgeschakeld, de Z80 negeert de interrupts van bijvoorbeeld de video processor.
Ook bestaat er een instruktie die het weer mogelijk maakt om de interrupt mode weer in te schakelen:

EI

EI, staat voor Enable Interrupt. Nadat deze instruktie is gegeven, is het weer mogelijk voor de Z80 om interrups 'te zien'.

Tot nu toe hebben we alleen nog maar gebruik gemaakt van het geheugen en de registers. Een MSX computer biedt echter meer, zoals de video chip en de geluids chip. Met de instrukties die hieronder volgen kunnen we dit soort chips besturen.

De Z80 staat in verbinding met een aantal externe 'apparaten', zoals bijvoorbeeld de video processor. Het is daardoor mogelijk deze apparaten te besturen via de Z80. Dit gebeurt via de input/outputpoort.(I/O)
In BASIC gebeurt het zenden van data naar een outputpoort (een randapparaat) m.b.v. het 'OUT' statement. Het ontvangen van data verloopt via het 'INP' statement.
In de assembler taal zijn deze instrukties ook aanwezig.

Voorbeelden:
Om het gegeven dat register A bevat naar

poort 99H te sturen, dient de instruktie 'OUT (099H),A' gegeven te worden. Om een gegeven, dat op poort 98H staat te wachten in het A register te laden, dient de instruktie 'IN A,(098H)' gegeven te worden.

Bij MSX-computers is het echter gevaarlijk om zelf de 'OUT' of 'IN' instrukties te gebruiken omdat sommige poorten in de toekomst veranderd kunnen worden. Waardoor het programma niet meer goed zal kunnen functioneren.

Daarom zullen wij in dit boek deze instrukties ook niet direct gebruiken.
Wij zullen gebruik maken van ROM routines, dit zijn subprogramma's die in de BASIC ROM zitten. Ze bevatten altijd de juiste informatie over de output- en input poorten.

Romroutines en het gebruik

De gebruikte routine in het eerste programma schrijft een byte in het Video RAM. Register paar HL moet het adres bevattten, het A register de te schrijven byte. Vergelijk met VPOKE HL,A.

Programma:

```
1 ORG 9000H
2 LD HL,01H  ;VRAM adres
3 LD A,50H   ;De te schrijven byte
4 CALL 04DH  ;Schrijf A naar VRAM  adres HL
5 RET
6 END
```

Uitleg:

Regel 1: Dit is een gegeven voor de assembler, de machinetaal wordt vanaf adres 9000H geladen.

Regel 4: Dit is een nieuwe instruktie. De algemene vorm is:

CALL NN

NN staat voor een 16 bits adres.
Deze instruktie kunt u vergelijken met 'GOSUB' in BASIC. Bij het aanroepen van deze instruktie wordt de inhoud van het PC register op de stapel gezet, vervolgens wordt het adres NN in het PC register geladen. Als de Z80 een RET instruktie

tegenkomt, dan wordt het PC register weer geladen met de twee bovenste bytes van de stapel, meestal zijn dit dezelfde bytes die door de instruktie op de stapel geplaatst waren.
De instruktie bewerktstelligt dus een sprong van het hoofdprogramma naar het subprogramma dat begint op adres NN. Bij een RET wordt er weer terug naar het hoofd programma gesprongen.
In dit geval wordt er naar de subroutine op adres 04DH gesprongen. Dit adres ligt in de BASIC ROM. Deze routine zorgt ervoor dat de byte die het A register bevat op Video RAM locatie HL wordt geladen.

Vraag: Waarom staat er in regel 5 'RET', dit programma is toch geen subroutine?

Antwoord: Dit programma is wel degelijk een subroutine. Bij het aanschakelen van de computer komt u automatisch in het operating system (de BASIC) terecht, dit is het hoofdprogramma. Bij het aanroepen van een machinetaal programma springt u dus uit het hoofdprogramma (de BASIC interpreter). Als u weer naar het hoofdprogramma terug wilt, dan moet u een RET geven.

Het programma vertaald zodat het onder BASIC in het geheugen geladen kan worden:

```
10 DATA 21,01,00,3E,50,CD,4D,00,C9
20 CLEAR 200,&H8FFF
30 FOR X=&H9000 TO &H9008
40 READ A$:POKE X,VAL("&H"+A$)
50 NEXT
60 DEFUSR=&H9000
```

Na het 'runnen' van dit programma en het ingeven van X=USR(0), bevat VRAM locatie 01H de waarde 50H, links boven in beeld verschijnt de hoofdletter 'P'. Het doel van dit programma kan ook gerealiseerd worden met 'VPOKE 1,&H50'.

Met het volgende programma is het mogelijk een buitenrand (border) in screen0 te laten verschijnen.
Het programma zet elk karakter om in z'n inverse, daardoor onstaat er een buitenrand.
Het programma bestaat uit 3 delen en werkt als volgt:

Eerst wordt de gehele patroon generator tabel, die begint op VRAM adres 800H, naar het RAM gecopieert (deel A). Daarna wordt elk gecopieerde byte geinverteerd (deel B). Vervolgens worden alle veranderde bytes weer op hun plaats in het VRAM terug gezet (deel C).
Er wordt gebruik gemaakt van de volgende ROM routines:

059H - Deze routine copieert een gespecificeerd blok bytes van het VRAM naar het RAM. Het eerste adres van het te copieren blok in het VRAM dient in registerpaar HL geladen te worden. Het eerste RAM adres dient in DE te staan. De lengte van het blok dient in BC staan.

05CH - Deze routine copieert een blok bytes van het RAM naar het VRAM. In HL dient het RAM adres van het te

copieren blok te staan (source), in DE het VRAM adres (destination) en in BC de lengte van het blok.

Het programma

```
;DEEL A

ORG 9000H
DI
LD HL,01800H      ;(VRAM) SOURCE
LD DE,0A000H      ;(RAM) DESTINATION
LD BC,0800H       ;LENGTE
CALL 059H         ;BLOK VAN RAM NAAR VRAM

;DEEL B

LD BC,0800H       ;AANTAL  BYTES  DAT
                  ;VERANDERT MOET WORDEN
LD DE,0A00H       ;VANAF HIER  STAAN  DE
                  ;BYTES
LOOP: LD A,(DE)   ;PAK BYTE IN A
XOR 0FFH          ;INVERTEER HEM
LD (DE),A         ;ZET BYTE TERUG
INC DE            ;POINTER  NAAR  VOLGENDE
                  ;BYTE
DEC BC            ;NOG BC BYTES TE GAAN
LD A,B            ;LAAD A MET B
OR 00H            ;KIJK OF A,B=0
JR NZ,LOOP        ;NEE, GA DAN NAAR LOOP

;DEEL C

LD HL,0A000H      ;BEGIN VAN VERANDERDE SET
LD DE,01800H      ;HIER MOET SET HEEN
LD BC,0800H       ;LENGTE VAN DE SET
CALL 05CH         ;ZET SET TERUG
EI                ;INTERRUPT AAN
RET               ;TERUG NAAR BASIC
```

```
23 END              ;EINDE
```

Programma vertaald naar BASIC;

```
10 DATA f3,21,00,08,11,00,A0,01,00
20 DATA 08,CD,59,00,01,00,08,11,00
30 DATA A0,1A,EE,FF,12,13,0B,78,F6
40 DATA 00,20,F5,21,00,A0,11,00,08
50 DATA 01,00,08,CD,5C,00,FB,C9,**
60 CLEAR 200,&H8FFF
70 T=&H9000
80 READ A$:IF A$="**" THEN 90 ELSE PO
KE T,VAL("&H"+A$):T=T+1:GOTO 80
90 SCREEN 0:DEFUSR=&H9000
```

Na x=usr(0) is de gehele karakter set veranderd in z'n inverse. Het lijkt dan net alsof er een border (rand) aan de buitenkant van het scherm is ontstaan.
Als u nog een keer x=usr(0) ingeeft, dan verandert de inverse karakter set weer in de normale.

Hooks

De BASIC ROM bevat CALL instrukties die naar bepaalde plaatsen in het RAM springen. Gewoonlijk staat er op zo'n plaats een RET instruktie (C9H), er wordt dan dus weer terug naar de ROM gesprongen. Het is echter mogelijk om een andere instruktie op die plaats te zetten, zodat deze eerst uitgevoerd wordt. Bovengenoemde plaatsen in het RAM worden ook wel HOOKS genoemd. Een HOOK neemt 5 geheugen plaatsen in beslag.
Dus een HOOK is een set van 5 geheugen-locaties in het RAM die via het ROM door een call instruktie worden aangesproken.
Geheugen locatie 1: Dit is de geheugen locatie die wordt aangesproken via de CALL instruktie.

Geheugen locaties 2 t/m 5: Deze volgen direct op geheugen locatie 1.

Een voorbeeld van een hook is geheugen-locatie FDC2H. (Geheugenplaatsen FDC3H t/m FDC7H horen ook bij deze hook)
Elke keer als er een toets ingedrukt wordt, wordt er naar dit adres gesprongen. Hiervoor zorgt de interrupt routine die op adres 038H begint (zie interrups). Normaal is deze geheugen locatie met C9H geladen, oftewel RET. Echter we kunnen ook een andere instruktie op deze locatie zetten (We kunnen de instruktie maximaal 5 bytes groot maken omdat we vijf geheugen-

locaties tot onze beschikking hebben namelijk FDC2H t/m FDC6H.
Hier volgt een voorbeeld.

Tik in : poke &hfdc3,&hc0
         poke &hfdc4,&h00
         poke &hfdc2,&hcd

Wat hebben we nu gedaan?

Op geheugen locatie FDC2H staat een CDH, dit betekent voor de Z80 'CALL NN'. Hierbij staat NN voor een 16 bits adres.
Geheugen locaties FDC3H en FDC4H vormen samen dit adres 00CDH. Dus de drie geheugenlocaties bevatten een instruktie die de subroutine (in dit geval een ROM routine) 00CDH aanroept. Deze routine laat een 'beep' klinken.
Als er nu een toets wordt ingedrukt, dan klinkt er een 'beep'.

Vraag: Waarom worden eerst de adressen FDC3H en FDC4H van een waarde voorzien en dan pas FDC2H?

Antwoord: Als we eerst adres FDC2H met de waarde CDH laden, dan staat er geen C9H (RET) meer op deze locatie. Dus dan wordt de CALL NN instruktie direct uitgevoerd echter geheugens FDC3H en FDC4H bevatten nog niet de juiste waarden. Het is dus niet bekend waar er naar toe wordt gesprongen.

Opdracht: Laad en 'run' eerst het vorige programma en wijzig de HOOK zodat, telkens bij het indrukken van een toets, de karakterset inverse wordt gemaakt.

```
Antwoord: poke &hfdc2,&hc9
          poke &hfdc3,&h00
          poke &hfdc4,&h90
          poke &hfdc2,&hcd
```

U moet nu begrijpen waarom adres FDC2H eerst met de waarde C9H (RET) geladen wordt.

Een lijst van de beschikbare hooks bij MSX kunt u vinden in appendix

Uitwisselen van registers

In BASIC bestaat het commando SWAP. Dit commando geeft de mogelijkheid om twee stringvariabelen uit te wisselen.
Voorbeeld:

```
         A$="MSX"
         B$="SVI"
    SWAP A$,B$
```

A$ bevat nu de inhoud van B$(SVI) en B$ de inhoud van A$(MSX).

Eenzelfde commando staat ons ter beschikking in machinetaal. Dit commando in assembler:

```
EX rgr,rgr
```

- rgr = IX, IY, BC, DE, HL, SP of AF

Voorbeeld: EX HL,DE

Dit wisselt de inhoud van HL en DE met elkaar om.

Als het EX commando wordt gebruikt bij indirecte adressering, dan wordt de inhoud van IX,IY of HL met de bovenste stapel eenheid uitgewisseld.

```
EX (SP),IX
EX (SP),IY
EX (SP),HL
```

Geindexeerd adresseren

Het geindexeerd adresseren is voor sommige toepassingen heel handig. De registers die voor deze voorziening worden gebruikt heten 'IX' en 'IY'. De beide registers zijn 16 bits groot. Het is niet mogelijk om deze te splitsen in twee 8 bit registers.

Het gemak van het geindexeerd adresseren komt vooral naar voren als we een reeks van data (gegevens) moeten inlezen. Als uitgangspunt nemen we een bepaald base-adres. Dit adres dient als markeringspunt van onze datareeks.
De vorm van de instrukties is de volgende:

```
LD A,(IX+d)

LD A,(IY+d)
```

De letter d, wordt het displacement byte genoemd. Deze byte wordt opgeteld bij het adres dat zich in het indexregister bevindt. Om het geheel iets duidelijker te maken zullen we nu een voorbeeld geven;

Wilt u uit een reeks van data die in het geheugen liggen opgeslagen een bepaalde byte ophalen, dan kunt u dat op de volgende wijze bewerkstelligen......

Stel dat u uw datatabel heeft opgeslagen vanaf 8200H en u wilt de 30e byte uit de

tabel inlezen, dan kunt dat op de volgende wijze doen.

```
LD IX,08200H
LD A,(IX+29)
```

We tellen dus 29 bij 8200H op voor de 30e byte. De eerste byte bevindt zich nl. op 8200H, dus ligt de 30e byte 29 plaatsen verder, vandaar die 29.

Het geindexeerd adresseren zult u veel tegenkomen in grote machinetaal programma's, maar u zult het in kleine routines vrijwel nooit gebruiken.

De stapel en bevelen

Net als bij het gebruik in BASIC, hebben we nu weer te maken met de stapel. Deze stapel bevindt zich in het RAM geheugen. Bij het programmeren in BASIC hoefden we ons eigenlijk nooit zorgen te maken om deze stapel. Bij het direct programmeren in machinetaal heeft u echter wel ter dege rekening te houden met de stapel. Gebruikt u echter een assembler, dan hoeft u zich om bepaalde onderdelen van de stapel geen zorgen te maken. Wat echter voor zowel de machinetaal alswel de assembler-programmeur van belang is, zijn de stapel-commando's.

De stapel wordt, zoals u reeds in de inleiding heeft kunnen lezen door uw MSX gebruikt om ervoor te zorgen dat tijdens een sprong instrukties geen geheugenverlies wordt geleden. In machinetaal bestaat bijvoorbeeld de instruktie CALL. Deze instruktie maakt het mogelijk om ROM- of eigen routines aan te roepen. Als u d.m.v. deze CALL-opdracht een routine aanspreekt wordt er dus een sprong gemaakt naar een ander deel in het geheugen. Het laatst doorlopen adres van het hoofdprogramma +1 wordt dan op de stapel geplaatst. Aan het eind van de routine staat er een RET (return) opdracht waardoor de Z80 weet dat dit het einde van de routine is. Op dat moment wordt het adres van de stapel gehaald. Nu loopt het hoofdprogramma dus

zonder problemen verder.

De stapel ligt in de top van het geheugen en groeit, als er meer waarden op worden geplaatst in het geheugen naar beneden. In de Z80 is er een register dat altijd naar de laatste waarde in de stapel verwijst. Dit register wordt het SP-register genoemd. (SP = Stack/Stapel Pointer) De plaats van de stapel is ondermeer ook afhankelijk van de bevelen CLEAR en MAXFILES.

Een voorbeeld van een mogelijke stapelindeling vindt u hieronder;

E500H - Vorige invoer
E4FFH - Vorige invoer
E4FEH - Vorige invoer
E4FDH - Vorige invoer
E4FCH - Laatse invoer
E4FBH - Nog in te voeren

De Stapel Pointer zal bij de boven afgebeelde stapel dus de waarde E4FCH hebben.

Laadt u tijdens het verloop van een spronginstructie het SP-register met een andere waarde dan zal het programma na een CALL en RET terugkeren op deze andere waarde (verg. GOTO in BASIC). De instruktie om de inhoud van het SP-register te veranderen is:

```
LD SP,x
```

Gebruikt u deze instruktie echter alleen

als u zeker bent dat het programma na het gebruik ervan niet vastloopt.

Data op de stapel

Een heel belangrijke toepassing van de stapel is het gebruik als data opslagplaats. De instrukties die in dit verband worden gebruikt zijn:

PUSH - data naar de stapel

POP - data van de stapel

Bij het PUSH commando wordt de data weggeschreven nadat de SP met 1 is verlaagd. Bij het POP commando wordt de data van de stapel afgehaald en de SP met 1 verhoogd. Met de instrukties POP en PUSH kunnen er 16-bit register paren respectievelijk worden af- en opgestapeld.

De PUSH- en POP-instrukties hebben de volgende notatiewijze:

PUSH rgp	POP rgp
PUSH IX	POP IX
PUSH IY	POP IY

- rgp staat hier voor registerpaar. (BC, DE, HL EN AF)

Een van de meest voorkomende toepassingen van de PUSH en POP instrukties is bij het gebruik van ROM- of RAM-routines. Bij het gebruik van deze routines is het nl. vaak nodig om het HL-registerpaar met een be-

paalde waarde te laden. Als deze subroutine dan ook nog gebruik maakt van dit HL registerpaar, dan moet de inhoud van HL op de STAPEL worden weggeschreven, anders is de inhoud verloren. Met POP HL kunt u de waarde, nadat de routine is doorlopen, weer van de STAPEL afhalen. Ook de instrukties PUSH AF en POP AF komen redelijk vaak voor, omdat sommige routines de inhoud van de A of F registers wijzigen.

Het (DEF) USR commando

Het USR commando hebben we tot nog toe slechts gebruikt om onze machinetaal programma's te starten. We gaven dan met de opdracht DEFUSR=startadres het begin van ons programma aan en met X=USR(0) werd het programma gestart. Met USR zijn echter veel meer dingen mogelijk. Hierop zullen we zo terugkomen.

Het zou heel goed mogelijk kunnen zijn, dat u meerdere machinetaal routines vanuit uw BASIC programma's wilt oproepen. Elk machinetaal programma heeft dan een eigen startadres. Deze startadressen moet u in het BASIC programma definieren met het commando:

DEFUSR(X)=startadres.

Hierbij kan (X) de waarden 0 t/m 9 aannemen, zodat u in totaal 10 routines vanuit uw BASIC programma kunt aanroepen. Met X=USR(X)(0) wordt een programma dan gestart.

Zoals we reeds opmerkten, is het USR commando tot veel meer in staat dan alleen het starten van een machinetaal programma. Het is nl. mogelijk om vanuit een BASIC programma variabelen door te geven aan een machinetaal programma of vanuit een machinetaal programma aan BASIC.

A=USR(&H2000), zorgt ervoor dat de waarde

2000H ergens in het RAM geheugen wordt opgeslagen, terwijl tegelijkertijd de machinetaal routine wordt gestart.

Zoals reeds in de inleiding is vermeld onder het hoofdstuk 'variabelen', weet uw MSX computer met wat voor soort variabele hij te maken heeft door aan elke soort variabele een eigen waarde toe te kennen. Deze waarde wordt opgeslagen op adres F663H.

Met PRINT PEEK(&HF663) vindt u de inhoud van dit adres.

2 -	integer	variabele
3 -	string	variabele
4 -	enkelvoudige precisie	variabele
8 -	dubbele precisisie	variabele

We zullen nu een voorbeeld geven van het gene wat we tot nu toe hebben besproken. We moeten altijd beginnen met het definieren van het startadres. We nemen hiervoor nu:

```
10  DEFUSR3=&HC500
```

Dit zou bijvoorbeeld het startadres kunnen zijn van een programma dat telkens een bepaalde waarde nodig heeft om mee te kunnen werken. We kunnen deze waarde nu overbrengen d.m.v. het volgende commando:

```
20  X=USR3(&H0500)
```

Dit zorgt ervoor dat uw machinetaal programma zal worden gestart , dat als start-

adres C500H heeft. De constante 0500H zal worden opgeslagen op F7F8H en F7F9H. Stel dat deze waarde moest worden geladen in het HL-registerpaar, dan moet het programma het volgende doen:

```
LD HL,(0F7F8H)
```

Hierna zal HL de waarde 0500H bevatten. Om een waarde van een machinetaal programma over te brengen naar een BASIC programma, moet u het adres F7F8H laden met de door u gewenste waarde. Als u dan weer tegugkeert naar BASIC zal de variabele X de waarde bevatten dat zich op adres F7F8H bevindt.

Het overbrengen van strings is ietwat meer gecompliceerd. Echter met uw kennis van de variabelen opgedaan in de inleiding zal ook dit onderwerp geen grote moeilijkheden kunnen veroorzaken. De adressen F7F8H en F7F9H bevatten niet de string. Ze bevatten echter wel het adres met de verdere informatie over de string. Deze informatie bestaat uit de string-lengte (bytenr. 1), informatie waar de string is opgeslagen (bytenr. 2 en 3). Ook nu zullen we weer een voorbeeld geven ter verduidelijking van de besproken stof.

Voorbeeld:

```
10 A$="STARK"
20 B$=USR4(A$)
```

Het adres F663H bevat nu de waarde 3, omdat we hier met een string te maken hebben. De adressen F7F8H en F7F9H bevat-

ten nu het adres van de stringinformatie, bijvoorbeeld 8020H en 8021H de lengte (5) en 8022H en 8023H bevatten de locatie van de string.

- Enkelvoudige precisie variabelen worden opgeslagen op de adressen F7F6H-F7F9H.
- Dubbele precisie variabelen worden opgeslagen op de adressen F7F6H-F7FDH.

De SCROLL routine

Met MSX BASIC is het mogelijk om kleine delen van het beeld te laten bewegen. Deze bewegende delen worden sprites genoemd. Het is jammer genoeg niet mogelijk om het gehele grafische scherm te bewegen omdat de BASIC daarvoor te langzaam is. Echter m.b.v. machinetaal is dat wel mogelijk. De vraag is alleen hoe dat zou moeten. Onderstaande routine geeft een mogelijk antwoord op deze vraag.

```
ORG 0D000H
LD HL,01800H
LD DE,0D100H
LD BC,0300H
CALL 059H
SCROLL: LD DE,0D100H
LD HL,0D101H
LD BC,0001FH
LOOP: PUSH BC
LD A,(DE)
LDIR
LD (DE),A
INC DE
INC HL
POP BC
LD A,0D4H
CP H
JR NZ,LOOP
DI
LD HL,0D100H
LD DE,01800H
LD BC,0300H
```

```
23 CALL 05CH
24 EI
25 RET
26 END
```

Uitleg van het programma:

N.B. In dit boek komt de preciese uitleg van de video processor niet aan de orde. We geven zo nu en dan alleen een korte beschrijving van de processor.

Het principe van het programma berust hierop: Tijdens het initialiseren van screen mode 2, wordt de name table opgesplitst in drie blokken van elk 256 bytes. Elk blok bestaat uit 256 hokjes, genummerd van 0 t/m 255. Elk hokje bevat na de initialisatie een waarde die overeen komt met het nummer van het hokje. Zo bevat hokje 0 van het eerste blok de waarde 0, hokje 3 van het eerst blok de waarde 3, hokje 8 van het derde blok de waarde 8 enz. De waarde die in een hokje staat is bepalend voor hetgeen we op het scherm te zien krijgen. Als in hokje 0 van het eerste blok , dat de waarde 0 bevat, b.v. een streepje staat en we willen dat hokjes 1 t/m 255 ook zo'n streepje komt, dan moeten we deze hokjes met de waarde nul vullen omdat de waarde nul refereert naar de inhoud van hokje nul.
Bij deze routine wordt ook van dit principe gebruik gemaakt. We veranderen steeds de inhoud van de name table op een zodanige manier waardoor het lijkt alsof het beeld naar links verplaatst wordt. We doen dit op de volgende manier: Een reeks van 1FH blokjes vormen tezamen een regel van

256*8 pixels. Als we hokje 0 van ieder van de drie blokken met de waarde 1 vullen, dan komt de informatie waar hokje 1 naar verwijst, in de patronen- en kleuren tabel, in hokje 0 terecht. Dus de tekening die in hokje 1 stond komt in hokje 0. Doen we dit voor elk hokje van een regel, dan is de tekening die op die regel stond naar links verschoven. (opmerking: de nummerieke inhoud van hokje 0 komt in het laatste hokje van een regel terecht). Dit verplaatsen van nummerieke waarden in een regel voeren we voor iedere regel uit.

Resultaat: Het beeld is 8 pixels naar links verschoven.

Regels 2 t/m 5: Deze instrukties zorgen ervoor dat de gehele name tabel, 300H bytes lang, in het RAM wordt gecopieerd.

Regel 9: PUSH BC, deze instruktie zet het BC register paar op de stapel. Dit doen we omdat we de inhoud van BC (1FH) steeds weer nodig hebben. We konden in principe ook steeds de instruktie LD BC, 01FH geven, echter deze instruktie is langzamer.

Regel 10: LD A,(DE): Deze instruktie zorgt ervoor dat we de nummerieke inhoud van het eerste blokje van iedere regel veilig stellen om het later aan het eind van iedere regel te zetten.

Regel 11: LDIR: Deze instruktie verplaatst de inhoud van een regel naar links (want de inhoud van HL en DE verschillen 1 in waarde).

Regel 12: LD (DE),A: DE verwijst op dit ogenblik naar het eind van de zojuist verschoven regel. We plaatsen dus de nummerieke inhoud die het eerste blokje van een regel had in het laatste blokje van de regel. Na deze instruktie is er een gehele regel naar links gescrold.

Regels 13 en 14: Deze instrukties zorgen ervoor dat HL en DE naar een nieuwe regel verwijzen: DE verwijst naar het eerste blokje van de nieuwe regel, HL naar het tweede.

Regel 15: POP BC: Deze instruktie laadt de bovenste twee bytes van de stapel in BC. In dit geval de waarde 01FH, de lengte van een regel.

Regels 16 t/m 19: Met deze instrukties wordt bekeken of we het einde van de name tabel bereikt hebben. Dit is het geval als het H register de waarde 0D4H heeft bereikt (en het L register de waarde 00).
De instruktie CP H vergelijkt (CP staat voor compare) de inhoud van register H met de accumulator, bevatten ze dezelfde waarde dan wordt het ZERO bit in het FLAG register 0 gemaakt. Als het resultaat ongelijk aan 0 is dan hebben we het einde van de name tabel nog niet bereikt en dus moeten we het geheel nog een keer herhalen.

Regels 19 t/m 23 tenslotte, zorgen ervoor dat de veranderde name tabel in de VDP terecht komt. Na deze instrukties neemt u pas de verplaatsing van het beeld naar links waar.

Het BASIC poke programma:

```
10 DATA 21,00,18,11,00,D1,01,00,03,CD
20 DATA 59,00,11,00,D1,21,01,D1,01,1F,00
30 DATA C5,1A,ED,B0,12,13,23,C1,3E,D4,BC
40 DATA 20,F3,F3,21,00,D1,11,00,18,01,00
50 DATA 03,CD,5C,00,FB,C9,**
60 CLEAR 200,&HCFFF
70 T=&HD000
80 READ A$:IF A$="**" THEN 90 ELSE PO
KE T,VAL("&H"+A$):T=T+1:GOTO80
90 DEFUSR1=&HD000:DEFUSR2=&HD00C
```

Na het 'runnen' van dit programma, zit alle informatie voor het scrollen in het geheugen. Nu moet u het volgende BASIC programma intikken en runnen.

```
10 STOP ON:ON STOP GOSUB 180: COLOR 15,1,1:SCREEN2
:LINE (240,70)-(100,70):LINE-(100,30),15:LINE-(50,
80),15:LINE-(100,130),15:LINE-(100,95):LINE-(240,9
5):LINE-(240,70):PAINT(180,72),15
110 COLOR 1:OPEN"GRP:"AS1:PRESET (70,75):PRINT#1,"
MSX MACHINETAAL HAND-
120 PRESET(70,75):PRINT#1,"MSX MACHINETAAL HAND-":
PRESET(70,85):PRINT#1,"BOEK SCROLL-ROUTINE":X=USR1
(0):'ZET DE NAME TABLE IN HET GEHEUGEN.":X=USR2(0)
:'SCROLT HET GEHELE BEELD 8 PIXELS NAAR LINKS.
150 X=USR2(0):'SCROLT HET GEHELE BEELD 8 PIXELS NA
AR LINKS.
160 FORM=1TO55:NEXT:'PAUZE
170 GOTO 150
180 WIDTH40:COLOR 15,4,4:END
```

Als u alles goed gedaan heeft, dan ziet u de pijl naar links bewegen. I.p.v. de pijl kunt u natuurlijk ook andere figuren laten bewegen.

Tip: Door regel 22 in het assembler programma te vervangen door 'LD BC,0200H', scrolt slechts twee derde van het scherm naar links.

Cassette - diskette conversie

P.S.: DIT ONDERWERP IS SLECHTS BRUIKBAAR BIJ Z.G. 80K RAM MSX SYSTEMEN....

Dit onderwerp is een veel voorkomend vraagstuk bij de bezitters van een disk-drive. Deze mensen moeten, als zij software in de winkel kopen, het meestal doen met programma's op cassette. Dit medium leent zich zeer goed voor computer gebruik, omdat het zeer goedkoop kan worden geproduceerd, en de afspeelapparatuur bij elke MSX bezitter wel aanwezig zal zijn. Het nadeel is, zoals bij u zeker bekend, de lange wachttijden bij het laden/saven van een programma.

Om dit probleem bij de bezitter van een diskdrive uit de wereld te helpen, publiceren wij nu enkele gebruiksprogramma's om uw software van cassette naar diskette over te zetten.

Bij deze programma's maken wij gebruik van het z.g. 'BANKSWITCHING' en het 'LDIR' commando.

Voordat we met de programma bespreking beginnen, zullen we u eerst vertrouwd maken met het begrip bankswitching. Zoals u weet is de Z80 niet in staat om meer dan

64K aan geheugen (ROM/RAM) tegelijkertijd te adresseren. Uw computer (als dit een z.g. 80K machine is) beschikt echter over meer dan deze 64K geheugen. Dit geheugen bestaat nl. vaak uit 64K RAM, 32K ROM en 16K Video RAM. Het Video RAM wordt bestuurt door de Video chip, dus dit valt indirect buiten het adresserings bereik van de Z80. De overige 96K geheugen kan echter wel door de Z80 worden geadresseerd.

Het geheugen in uw MSX computer is verdeeld in z.g. banken van 16K. Deze banken kunnen dan in groepjes van 4 tegelijkertijd door de Z80 worden bestuurd (4*16=64). Een bank van 16K wordt in het computerjargon aangeduid als een pagina (page). Er zijn vier pagina's: Pagina 0 loopt van 0000H tot 4000H, pagina 1 van 4000H tot 8000H, pagina 2 van 8000H tot C000H en pagina 3 van C000H tot FFFFH. 64K geheugen is dus gelijk aan vier pagina's, oftewel om in MSX termen te spreken: een slot.

Hierna volgt een mogelijke indeling van de geheugenopbouw in slots/pages in dit geval die van de SpectraVideo SVI 728. Raadpleeg uw gebruiksaanwijzing voor de gehanteerde geheugenindeling bij uw computer. Bij de SONY HB75P is de ROM op de zelfde plaats gelegen, echter de 64K RAM zit nu in slot 2. Bij de PHILIPS VG 8020 zit de ROM in slot 0 en de 64K RAM in slot 3. Dit kan er de oorzaak van zijn dat sommige software niet op alle MSX computers loopt.

SVI-728 MEMORY LAYOUT:

	SLOT-0	SLOT-1	SLOT-2	SLOT-3
0000H pgn. 0	ROM (MSX BASIC)	16K RAM		
4000H pgn. 1	ROM (MSX BASIC)	16K RAM		
8000H pgn. 2		16K RAM		
C000H pgn. 3		16K RAM		
FFFFH	EXP. MOD. INTERFACE	USER RAM 64K	GAME SLOT	EXP. SLOT

Willen we nu in b.v. de adressen 0000H t/m 8000H (pagina 0 en pagina 1) de MSX ROM plaatsen, dan moeten we pagina 0 en pagina 1 uit slot 0 kiezen. Willen we verder op de adressen 8000H t/m FFFFH (pagina 2 en pagina 3) RAM plaatsen, dan moeten we pagina 2 en pagina 3 uit slot 1 kiezen.

Om terug te komen op ons allereerste doel, nl. het overzetten van cassette naar dis-

kette, zullen we nu een programma (PROGRAMMA A) aan u voorstellen dat ervoor zorgt dat het door u ingeladen machinetaal programma vanuit pagina 2 en 3 naar pagina 0 en 1 van slot 1 wordt gecopieerd.

P.S. Het programma dat we als voorbeeld van ons naar disk overzetten zullen gebruiken is het spel 'ROGER RUBBISH' (van SpectraVideo)

```
 1 ORG 0F200H
 2 DI
 3 LD A,01010101B
 4 OUT (0A8H),A
 5 LD HL,08000H
 6 LD DE,00000H
 7 LD BC,07200H
 8 LDIR
 9 LD A,01010000B
10 OUT (0A8H),A
11 EI
12 RET
13 END
```

Uitleg:

Regel 2: DI, schakelt de interrupt uit. Dit voorkomt dat de hardware interrupt naar adres 038H springt.

Regel 3: LD A,01010101B, 01010101B vormt de informatie voor de computer i.v.m. de pagina indeling. Zie verder uitleg regel 4.

Regel 4: OUT (0A8H),A: Deze instruktie stuurt de inhoud van register A naar poort

nr. A8H. Via deze poort kunnen we de gewenste geheugen indeling realiseren.
Verklaring van de inhoud van het A register: Door poort OA8H kunnen we 8 bits aan informatie versturen. De gewenste geheugen indeling ligt dus vast in deze 8 bits, en wel als volgt:

Bit 0 en 1: Deze bevatten de informatie omtrent het geselecteerde slot voor pagina nr. 0. In ons geval (SVI 728) selecteren we met de binaire waarde 01B (bit 0 en 1 van het A register) slot 1 voor pagina 0.

Bit 2 en 3: Deze bevatten de informatie omtrent het geselecteerde slot voor pagina nr. 1. In ons geval selecteren we met de binaire waarde 01B (bit 2 en 3 van het A register) wederom slot 1 voor pagina 1.

Bit 4 en 5: Deze bevatten de informatie omtrent het geselecteerde slot voor pagina nr. 2. Nu dus slot 1 voor pagina 2.

Bit 6 en 7: Logischerwijze staan deze bits voor de slot selectie van pagina 3, nl. nogmaals slot 1 voor pagina 3.

Als uiteindelijk resultaat hebben we dus het gehele 64K RAM geheugen voor ons eigen gebruik geselecteerd. De oplettende lezer zal nu tot de conclusie komen, dat de ROM met de BASIC interpreter niet meer gebruikt kan worden.

Regel 5: LD HL,08000H: Geeft het begin adres van het te copieren blok aan, dat we gaan verplaatsen met het LDIR statement.

Regel 6: LD DE,0000H: Geeft het doel adres (destination) van het te copieren blok aan.

Regel 7: LD BC,0720OH: Geeft de lengte van het te copieren blok aan.

Regel 8: LDIR: Copieert het aangegeven blok (we copieren dus adressen 8000H t/m F200H naar de adressen 0000H t/m 7200H in het RAM!!).

Regel 9: LD A,01010000B: Het A register bevat nu de gegevens omtrent het inschakelen van het ROM in pagina 0 & 1 en het RAM in pagina 2 & 3.

Regel 10: OUT (0A8H),A: Deze instruktie stuurt de inhoud van register A naar poort nr. A8H. Bit 0 & 1 selecteren slot 0 voor pagina 0. Bit 2 & 3 selecteren slot 0 voor pagina 1. Bit 4 & 5 selecteren slot 1 voor pagina 2. Bit 6 & 7 selecteren slot 1 voor pagina 3.

Regel 11: EI, schakelt de eerder uitgeschakelde interrupt weer aan.

Regel 12: RET: terug naar BASIC.

We zullen nu het BASIC programma laten zien, dat bovenstaand assemblerprogramma naar het geheugen overbrengt.

- PROGRAMMA A:

```
10 DATA F3,3E,55,D3,A8,21,00,80,11,00,00
20 DATA 01,00,72,ED,B0,3E,50,D3,A8,FB,C9
30 DATA **
```

```
40 CLEAR 25,&H8800
50 T=&HF200
60 READ A$:IF A$="**" THEN 70 ELSE POKE T,
VAL("&H"+A$):T=T+1:GOTO 60
70 DEFUSR=&HF200:NEW
```

Nu we het eerste programma uit onze reeks van vijf aan u hebben voorgesteld, kunnen we het gebruik ervan uitleggen.

De procedure om te volgen is deze:

- Laad de 'header' van het machinetaal programma dat u naar disk wilt overzetten in uw computer (met afgeschakelde disk-drive). De header is na ongeveer 10 sec. ingelezen. Nu dient u het z.g. startadres van het programma op te vragen met.

```
?HEX$((PEEK(&HFCBF))+256*(PEEK(&HFCC0)))
```

Dit adres dient u te onthouden.

- Start uw computer op met aangesloten diskdrive, maar MET DE SHIFT TOETS INGEDRUKT. Laad vervolgens programma A van een cassette in het geheugen en 'run' het. Nu kunt u het programma dat naar disk moet worden overgezet in het geheugen laden. Dit doet u met BLOAD"CAS:". Let op! Zet nooit de toevoeging ',R' achter de laad-instruktie, daar de computer het programma dan direct activeert na het laden. Na het inladen geeft u de opdracht: X=USR(0)
We hebben nu dus het programma naar de geheugen adressen 0-7200H in het RAM gecopieerd.
Als dit gebeurd is, dan moet u de computer opnieuw opstarten d.m.v. een reset. Als uw

computer geen reset schakelaar heeft, dan moet u het volgende intikken:

DEFUSR=0:X=USR(0) + (ENTER)

Let er wel op dat de drive aangesloten is en er zich geen diskette in de drive bevindt. Door de reset wordt de inhoud van 'de onzichtbare RAM' niet aangetast. De Disk BASIC is nu geactiveerd.

De volgende stap is het wegschrijven van het op disk te zetten programma in twee delen. De volgende twee programma's (B en C) bewerkstelligen dat.

Ten eerste (prg. B): Het copieren van het geheugen deel 0000H tot 4000H (RAM!) naar het geheugen deel 8800H tot C800H en het vervolgens wegschrijven van het 8800H tot C800H deel naar een diskette.

Ten tweede (prg. C): Het copieren van het geheugen deel 4000H tot 7200H (RAM!) wederom naar geheugen deel 8800H to C800H en het vervolgens wegschrijven van het 8800H tot C800H naar een diskette.

PROGRAMMA B:

```
 1 ORG 0D200H
 2 DI
 3 LD A,01010101B ; SLOT SELECTIE
 4 OUT (0A8H),A   ; REALISEER SLOTSELECTIE
 5 LD HL,0000H    ; BEGIN ADRES
 6 LD BC,08800H   ; DOEL ADRES
 7 LD BC,04000H   ; LENGTE
 8 LDIR           ; COPIEREN VAN BLOK
 9 LD A,01010000B ; SLOT SELECTIE
10 OUT (0A8H),A   ; REALISEER SLOTSELECTIE
```

```
11 EI
12 RET
13 END
```

Het BASIC poke programma ziet er nu alsvolgt uit:

```
10 DATA F3,3E,55,D3,A8,21,00,00,11,00,88
20 DATA 01,00,40,ED,B0,3E,50,D3,A8,FB,C9
30 DATA **
40 CLEAR 25,&H87FF
50 T=&HD000
60 READ A$:IF A$="**" THEN 70 ELSE POKE T,
VAL("&H"+A$):T=T+1:GOTO 60
70 DEFUSR=&HD000:NEW
```

Uitleg van het gebruik (pgr.B): Laad dit programma van disk. 'Run' het programma en tik in: X=USR(0) +(ENTER)
Nu staat het eerste deel van het programma dus op locaties 8800H tot C800H. Dit schrijft u nu weg met:

```
BSAVE"A:DEEL1.OBJ",&H8800,&HC800
```

PROGRAMMA C:
Dit programma is vrijwel identiek aan programma B. U dient nu alleen regel 5 te veranderen van LD HL,0000H in LD HL,04000H. In het BASIC-deel dient u de achtste byte (00) te veranderen in 40.

Uitleg van het gebruik (pgr.C): Laad dit programma van disk. 'Run' het programma en tik in: X=USR(0) +(ENTER)
Nu staat het tweede deel van het programma dus op locaties 8800H tot C800H. Dit schrijft u nu weg met:

```
BSAVE"A:DEEL2.OBJ",&H8800,&HC800
```

Nu staat het het programma in principe op disk. De kunst is nu het zo in het geheugen te plaatsen zodat we geen last hebben van de Disk BASIC. Daartoe heeft u nog 3 programma's nodig. De eerste twee programma's (D en E) zorgen ervoor dat het programma weer in de 'ontzichtbare RAM' komt te staan. Programma F tenslotte, zorgt ervoor dat alles op de juiste plaats komt.

PROGRAMMA D:

```
 1 ORG OD200H
 2 DI
 3 LD A,01010101B
 4 OUT (OA8H),A
 5 LD HL,08800H
 6 LD DE,0000H
 7 LD BC,04000H
 8 LDIR
 9 LD A,01010000B
10 OUT (OA8H),A
11 EI
12 RET
13 END
```

Het basic poke programma:

```
10 DATA F3,3E,55,D3,A8,21,00,88,11,00,00
20 DATA 01,00,40,ED,B0,3E,50,D3,A8,FB,C9
30 DATA **
40 CLEAR 25,&H87FF
50 T=&HD200
60 READ A$:IF A$="**" THEN 70 ELSE POKE T,
VAL("&H"+A$):T=T+1:GOTO 60
70 DEFUSR=&HD200:NEW
```

Instruktie bij het programma D: 'Run' dit programma. Laad 'DEEL1.OBJ' van disk. Tik nu in: X=USR(0) +(ENTER)

PROGRAMMA E:

Zie programma D, met dien ter verstande dat regel 6 veranderd moet worden van LD DE,0000H in LD DE,04000H en regel 7 van LD BC,04000H IN LD BC,3200H

In het BASIC programma dienen data nr. 11 en 14 vervangen te worden door respectievelijk 40 en 32

Instruktie bij het programma E: 'Run' dit programma. Laad 'DEEL2.OBJ' van disk. Tik nu in: X=USR(0) + (ENTER). Het gehele programma zit nu in de 'onzichtbare RAM'. We hoeven nu alleen nog het programma op de plaats waar het hoort te zetten.

PROGRAMMA F:

```
ORG 0FFE7H
DI
LD A,01010101B
OUT (0A8H),A
LD HL,0000H
LD DE,08000H
LD BC,07200H
LDIR
LD A,01010000B
OUT (0A8H),A
EI
RET
END
```

Basic poke programma:

```
10 DATA F3,3E,55,D3,A8,21,00,00,11,00,80
20 DATA 01,00,72,ED,B0,3E,50,D3,A8,FB,C9
30 DATA **
40 CLEAR 25,&H87FF
50 T=&HFFE7
60 READ A$:IF A$="**" THEN 70 ELSE POKE T,
VAL("&H"+A$):T=T+1:GOTO 60
70 DEFUSR=&HFFE7:NEW
```

Het gebruik van progr. F: Laad het programma van disk. 'Run' het en tik hierna in: X=USR(0) en daarna;

DEFUSR=STRT.ADR.:X=USR(0)

Het startadres is in ons geval gelijk aan C000H.

Sommige machines lopen bij het uitvoeren van programma F vast. Hierdoor is het niet mogelijk om het machinetaalprogramma op te starten. We moeten in zo'n geval het programma F zo aanpassen dat het te starten programma daaruit wordt opgestart. Dit wordt gedaan d.m.v. een 'jump' naar het start adres. De syntax van deze instructie is:

JP startadres (C000H)

Zie hieronder het programma F in aangepaste versie;

```
1 ' DIT PROGRAMMA COPIEERT HET
2 ' NIET GEBRUIKTE 32K RAM NAAR
3 ' DE BOVENSTE 32K RAM ....
4 '
9 :
10 :
11  DATA f3       :'di
```

```
20 DATA 3E,55     :'LD A,01010101B
30 DATA D3,A8     :'OUT (A8H),A
40 DATA 21,00,00 :'ld hl,0000H
50 DATA 11,00,80 :'ld de,8000H
60 DATA 01,00,72 :'LD BC,7200H
70 DATA ED,B0     :'LDIR
110 :
120 :
130 :
140 :
150 DATA 3E,50    :'LD A,50H
160 DATA D3,A8    :'OUT (A8H),A
174 DATA c3,00,c0:'JP c000
180 DATA **
190 CLEAR25,&H8800
200 T=&HF200
210 READ A$:IF A$="**" THEN 300 ELSE P
OKET,VAL("&H"+A$):T=T+1:GOTO 210
300 DEFUSR=&HF200:X=USR(0)
```

Appendix A
Een disassembler

Naast een assembler beschikt de gevorderde machinetaalprogrammeur meestal over een disassembler. Zo'n disassembler doet het tegenovergestelde van een assembler.

Het programma zorgt ervoor dat vanaf een opgegeven adres alle data in het geheugen wordt vertaald in begrijpelijke assembler instrukties. U kunt nu dus eventueel door andere mensen geschreven machinetaal programma's disassembleren, zodat u kunt kijken hoe zij hun programma's hebben gemaakt. Een voor de hand liggend programma is de in uw MSX aanwezige 32K ROM. Hiervan kunt u zeer veel leren.

Het programma is zeer makkelijk in het gebruik. U hoeft slechts het adres op te geven vanaf waar u wilt gaan disassembleren.

```
5 CLEAR 1000,&HD500
10 DEFINTA-Z
20 GOSUB780
30 COLOR1,14,14:SCREEN0:WIDTH40:KEYOFF:D
EFSNGA-Z:DIM A$(255),CB$(30),R$(7),ED$(5
4),EX(54)
40 FORX=0TO255:READA$(X):NEXT:FORX=0TO30
:READCB$(X):NEXT:FORX=0TO7:READR$(X):NEX
T:FORX=0TO54:READED$(X):NEXT:FORX=0TO54·
READEX$:EX(X)=VAL("&H"+EX$):NEXT
50 X=0:GOSUB530
```

```
60 ONABS(STRIG(0))GOSUB530:ONTDGOTO170:P
C=PC+1:PC=PC*-(PC<22)
70 P=PEEK(X):PA$=CHR$(P)
80 P$=A$(P):ML=X:LL=4:GOSUB520:MT$=ML$:M
L=P:LL=2:GOSUB520:MT$=MT$+":"+ML$
90 V=VAL(RIGHT$(P$,1)):ONSGN(V)GOSUB220
100 GOSUB420:GOSUB470:ONPPGOTO130
110 PRINTMT$TAB(16)P$TAB(33)PX$:ONPQGOTO
150
120 PRINTMT$TAB(16)P$TAB(33)PX$:GOTO150
130 PRINTMT$;:PRINTTAB(16)USINGP$;VV$;:P
RINTTAB(33)PX$:ONPQGOTO150
140 PRINTMT$;:PRINTTAB(16)USINGP$;VV$;:P
RINTTAB(33)PX$
150 X=X+1:ON PFGOTO60:ONSGN(PC)GOTO60
160 IFINKEY$=CR$GOTO60ELSEGOTO160
170 ONABS(STRIG(0))GOSUB550:ONTD+1GOTO60
,180
180 ML=X:LL=4:GOSUB520:MT$=ML$+":":XX=XX
-1:PA$="":FORTL=0TO31
190 X=X+1:P=PEEK(X):PA$=PA$+CHR$(P)
200 NEXT:GOSUB 470:PRINTMT$PX$
210 GOTO 170
220 P$=LEFT$(P$,LEN(P$)-1):ONVGOSUB 230,
250,260,270,290,300,320:RETURN
230 GOSUB240:P$=P$+ML$:RETURN
240 X=X+1:ML=PEEK(X):PA$=PA$+CHR$(ML):LL
=2:GOSUB520:MT$=MT$+ML$:RETURN
250 GOSUB240:MX=ML:GOSUB240:ML=ML*256+MX
:LL=4:GOSUB520:P$=P$+ML$:RETURN
260 GOSUB240:ML=ML+(ML>127)*256:ML=ML+1+
X:LL=4:GOSUB520:P$=P$+ML$:RETURN
270 GOSUB 240:M1=ML\8:M2=MLMOD8:P1$=CB$(
M1)+"    ":P2$=R$(M2):IFM1<7THEN P$=P1$+P
2$:RETURN
280 M3=(M1)MOD8:P$=P1$+HEX$(M3)+","+P2$:
RETURN
```

```
290 QQ$="IX":GOSUB330:RETURN
300 GOSUB240:XX=0:FORTL=1TO54:IFML=EX(TL
)THENXX=TL
310 NEXT:P$=ED$(XX):RETURN
320 QQ$="IY":GOSUB330:RETURN
330 GOSUB240:M4=ML:OS$="":P$=A$(ML):GOSU
B400:V=VAL(RIGHT$(P$,1)):R1=INSTR(P$,"(H
L)"):R2=INSTR(P$,"HL")
340 IFR1THENGOSUB240:OS$="+"+HEX$(ML)
350 IFVTHENGOSUB410
360 IFR1=0ANDR2THENMID$(P$,INSTR(P$,"HL"
),2)=QQ$:GOTO380
370 IFR1THENR3=LEN(P$):P1$=LEFT$(P$,R1-1
):P2$=MID$(P$,R1+4,R3):P$=P1$+"("+QQ$+OS
$+")"+P2$
380 RETURN
390 P$="ONBEKEND":RETURN
400 IF M4=0THENP$="ADD   HL,BC":RETURNEL
SERETURN
410 P$=LEFT$(P$,LEN(P$)-1):ONVGOSUB 230,
250,260,270,390,390,390:RETURN
420  PP=0
430 IFINSTR(P$,"\  \")THENGOSUB460:PP=1:
RETURN
440 IFINSTR(P$,"\\")THENGOSUB240:LL=2:GO
SUB520:VV$=ML$:PP=1:RETURN
450 RETURN
460 LL=2:FORVD=0TO1:X=X+1:ML=PEEK(X):GOS
UB520:MT$=MT$+ML$:VK(VD)=ML:PA$=PA$+CHR$
(ML):NEXT:ML=VK(1)*256+VK(0):LL=4:GOSUB5
20:VV$=ML$:RETURN
470 PX$="":ONASGOTO490:FOR SS=1TOLEN(PA$
):IFMID$(PA$,SS,1)<" "THENMID$(PA$,SS,1)
="»"
480 NEXT:PX$=PA$:RETURN
490 FORSS=1TOLEN(PA$):JJ=ASC(MID$(PA$,SS
```

```
,1)):JJ=(JJAND127):IFJJ<32THENJJ=175
500 PX$=PX$+CHR$(JJ):NEXT:RETURN
510 RETURN
520 DD$=HEX$(ML):ML$=STRING$(LL-LEN(DD$)
,48)+DD$:RETURN
530 PC=1:DEFUSR=&H468:X=USR(0):INPUT"Beg
in adres:";S$:IFRIGHT$(S$,1)="h"ORRIGHT$
(S$,1)="H"THENX=VAL("&h"+S$)ELSEX=VAL(S$
)
540 RETURN
550 GOSUB530:RETURN
560 DATA NOP,"LD    BC,2","LD    (BC),A",
INC    BC,INC    B,DEC    B,"LD    B,1","RC
LA","EX    AF,AF'","ADD    HL,BC","LD
A,(BC)"
570 DATA DEC    BC,INC    C,DEC    C,"LD
 C,1",RRCA,DJNZ   3,"LD    DE,2","LD    (D
E),A",INC    DE,INC    D,DEC    D
580 DATA "LD    D,1",RLA,JR    3,"ADD
HL,DE","LD    A,(DE)",DEC    DE,INC    E,D
EC    E,"LD    E,1",RRA,"JR    NZ,3","LD
   HL,2","LD    (\  \),HL",INC    HL,INC
 H
590 DATA DEC    H,"LD    H,1",DAA,"JR
Z,3","ADD    HL,HL","LD    HL,(\  \)",DEC
   HL,INC    L,DEC    L,"LD    L,1",CPL,"J
R    NC,3","LD    SP,2","LD    (\  \),A",
INC    SP,INC    (HL),DEC    (HL)
600 DATA "LD    (HL),1",SCF,"JR    C,3","
ADD    HL,SP","LD    A,(\  \)",DEC    SP,I
NC    A,DEC    A,"LD    A,1",CCF,"LD    B,
B","LD    B,C","LD    B,D","LD    B,E","
LD    B,H","LD    B,L","LD    B,(HL)"
610 DATA "LD    B,A","LD    C,B","LD
C,C","LD    C,D","LD    C,E","LD    C,H"
,"LD    C,L","LD    C,(HL)","LD    C,A",
```

```
"LD      D,B","LD      D,C","LD      D,D","LD
   D,E","LD      D,H"
620 DATA "LD      D,L","LD      D,(HL)","LD
   D,A","LD      E,B","LD      E,C","LD      E
,D","LD      E,E","LD      E,H","LD      E,L",
"LD      E,(HL)","LD      E,A","LD      H,B","
LD      H,C","LD      H,D"
630 DATA "LD      H,E","LD      H,H","LD
H,L","LD      H,(HL)","LD      H,A","LD      L
,B","LD      L,C","LD      L,D","LD      L,E",
"LD      L,H","LD      L,L","LD      L,(HL)","
LD      L,A","LD      (HL),B"
640 DATA "LD      (HL),C","LD      (HL),D","LD
   (HL),E","LD      (HL),H","LD      (HL),L",H
ALT,"LD      (HL),A","LD      A,B","LD      A,C
","LD      A,D","LD      A,E","LD      A,H","L
D      A,L","LD      A,(HL)"
650 DATA "LD      A,A","ADD     A,B","ADD
A,C","ADD     A,D","ADD     A,E","ADD     A,H"
,"ADD     A,L","ADD     A,(HL)","ADD     A,A",
"ADC     A,B","ADC     A,C","ADC     A,D","ADC
   A,E","ADC     A,H"
660 DATA "ADC     A,L","ADC     A,(HL)","ADC
   A,A",SUB     B,SUB     C,SUB     D,SUB     E,
SUB     H,SUB     L,SUB     (HL),SUB     A,"SBC
 A,B","SBC     A,C","SBC     A,D","SBC     A,E
","SBC     A,H","SBC     A,L","SBC     A,(HL)"
,"SBC     A,A
670 DATA AND     B,AND     C,AND     D,AND     E
,AND     H,AND     L,AND     (HL),AND     A,XOR
 B,XOR     C,XOR     D,XOR     E,XOR     H,XOR
 L,XOR     (HL),XOR     A,OR      B,OR      C,OR
   D,OR      E,OR      H,OR      L,OR      (HL),O
R      A
680 DATA CP      B,CP      C,CP      D,CP      E
,CP      H,CP      L,CP      (HL),CP      A,RET
```

```
NZ,POP    BC,"JP     NZ,2",JP     2,"CALL
NZ,2",PUSH   BC,"ADD    A,1",RST     0,RET
   Z,RET
690 DATA "JP     Z,2",4,"CALL   Z,2",CALL
 2,"ADC    A,1",RST    8,RET    NC,POP    DE
,"JP     NC,2","OUT   (\\),A","CALL   NC,2"
,PUSH   DE,SUB    1,RST   10H,RET    C,EXX,"
JP     C,2"
700 DATA "IN     A,(\\)","CALL   C,2",5,"S
BC    A,1",RST    18H,RET    PO,POP    HL,"J
P     PO,2","EX    (SP),HL","CALL   PO,2",P
USH   HL,AND    1,RST    20H,RET    PE,JP
(HL),"JP     PE,2"
710 DATA "EX     DE,HL","CALL   PE,2",6,XO
R    1,RST    28H,RET    P,POP    AF,"JP
P,2",DI,"CALL   P,2",PUSH   AF,OR     1,RST
   30H,RET    M,"LD      SP,HL","JP     M,2"
,EI,"CALL   M,2",7,CP      1,RST    38H
720 DATA RLC,RRC,RL  ,RR  ,SLA,SRA,SRL,BIT
,BIT,BIT,BIT,BIT,BIT,BIT,BIT,RES,RES,RES
,RES,RES,RES,RES,RES,SET,SET,SET,SET,SET
,SET,SET,SET,B,C,D,E,H,L,(HL),A
730 DATA Unknown,"IN     B,(C)","OUT   (C)
,B","SBC    HL,BC","LD     (\   \),BC",NEG,R
ETN,IM     0,"LD      I,A","IN     C,(C)","O
UT   (C),C","ADC    HL,BC","LD      BC,(\   \
)",RETI
740 DATA "IN     D,(C)","OUT     (C),D","SB
C    HL,DE","LD     (\   \),DE",IM     1,"LD
   A,I","IN     E,(C)","OUT   (C),E","ADC
  HL,DE","LD      DE,(\   \)",IM      2
750 DATA "IN     H,(C)","OUT    (C),H","SBC
   HL,HL",RRD,"IN      L,(C)","OUT   (C),L
","ADC    HL,HL",RLD,"SBC    HL,SP","LD
(\   \),SP","IN     A,(C)","OUT   (C),A","A
DC    HL,SP","LD     SP,(\   \)
```

```
760 DATA LDI,CPI,INI,OUTI,LDD,CPD,IND,OU
TD,LDIR,CPIR,INIR,OTIR,LDDR,CPDR,INDR,OT
DR
770 DATA 0,40,41,42,43,44,45,46,47,48,49
,4A,4B,4D,50,51,52,53,56,57,58,59,5A,5B,
5E,60,61,62,67,68,69,6A,6F,72,73,78,79,7
A,7B,A0,A1,A2,A3,A8,A9,AA,AB,B0,B1,B2,B3
,B8,B9,BA,BB
780 COLOR 1,14:SCREEN0:WIDTH40:KEYOFF:PR
INT"*************************************
***"
790 :FORZ=1TO4:PRINT"*
                      *":NEXT:PRINT"*******
*******************************"
800 LOCATE10,3,0:PRINT"Z80 DISASSEMBLER"
810 LOCATE 0,6:PRINT:PRINT"Met dit progr
amma kunt u elk gewenst":PRINT"deel van
geheugen disassembleren":PRINT:PRINT"BAS
IC programma's kunt u vanzelf-":PRINT"sp
rekend niet disassembleren.
815 PRINT:PRINT"Geef het startadres hexa
decimaal op.
816 PRINT:PRINT"*************************
***************"
820 MM$="Druk op ENTER/RETURN voor verde
r...
830 X=1:Y=22:FOR I=1TO LEN(MM$):LOCATE X
,Y:PRINTMID$(MM$,I,1);:X=X+1:BEEP:FORTZ=
1TO100:NEXT:NEXT
840 IFINKEY$=CHR$(13)THENSCREEN0:RETURNE
LSEGOTO840
```

Hooks

De hooks bevinden zich bovenin het RAM van adres FD9AH t/m FFC9H.

ADRES(HEX)	ROM-CALL	BESCHRIJVING
FD9AH	0C4AH	Interrupt
FD9FH	0C53H	Interrupt
FDA4H	08C0H	CHPUT routine
FDA9H	09E6H	Cursor aan
FDAEH	0A33H	Cursor uit
FDB3H	0B2BH	DSPFNK routine
FDB8H	0B15H	ERAFND routine
FDBDH	0842H	TOTEXT routine
FDC2H	10CEH	CHGET routine
FDC7H	071EH	Karakterset van ROM naar VRAM
FDCCH	1025H	Toetsenbord decoder
FDD1H	0F10H	Toetsenbord decoder
FDD6H	1398H	NMI routine
FDDBH	23BFH	PINLIN routine
FDE0H	23CCH	QINLIN routine
FDE5H	23D5H	INLIN routine
FDEAH	7810H	"ON DEVICE GOSUB"
FDEFH	7C16H	"DSKO$"
FDF4H	7C1BH	"SET"
FDF9H	FC20H	"NAME"
FDFEH	7C25H	"KILL"
FE03H	7C2AH	"IPL"
FE08H	7C2FH	"COPY"
FE0DH	7C34H	"CMD"
FE12H	7C39H	"DSKF"
FE17H	7C3EH	"DSKI$"
FE1CH	7C43H	"ATTR$"

FE21H	7C48H	"LSET"
FE26H	7C4DH	"RSET"
FE2BH	7C52H	"FIELD"
FE30H	7C57H	"MKI$"
FE35H	7C5CH	"MKS$"
FE3AH	7C61H	"MKD$"
FE3FH	7C66H	"CVI"
FE44H	7C6BH	"CVS"
FE49H	7C70H	"CVD"
FE4EH	6A93H	Locate FCB
FE53H	6AB3H	Locate FCB
FE58H	6AF6H	"OPEN"
FE5DH	6B0FH	"OPEN"
FE62H	6B3BH	Close I/O buffer 0
FE67H	6BB3H	"MERGE/LOAD"
FE6CH	6BA6H	"SAVE"
FE71H	6BCEH	"SAVE"
FE76H	6BD4H	"MERGE/LOAD"
FE7BH	6C2FH	"FILES"
FE80H	6C3BH	"GET/PUT"
FE85H	6C51H	Sequentiele output
FE8AH	6C79H	Sequentiele input
FE8FH	6CD8H	"INPUT$"
FE49H	6D03H	"LOC"
"	6D14H	"LOF"
"	6D25H	"EOF"
"	6D39H	"FPOS"
FE99H	6D0FH	"LOC"
FE9EH	6D20H	"LOF"
FEA3H	6D33H	"EOF"
FEA8H	6D43H	"FPOS"
FEADH	6E26H	"LINE INPUT£"
FEB2H	6F15H	Devicenaam benoeming
FEB7H	6F33H	Devicenaam benoeming
FEBCH	6F37H	Devicenaam benoeming
FEC1H		Heeft geen functie
FEC6H	6F8FH	I/O functie verzenden
FECBH	629AH	RUN-CLEAR

FED0H	62A1H	RUN-CLEAR
FED5H	62AFH	RUN-CLEAR
FEDAH	62F0H	Reset stapel
FEDFH	145FH	ISFLIO routine
FEE4H	1B46H	OUTDO routine
FEE9H	7328H	CR,LF naar OUTDO
FEEEH	7374H	Hoofdlus line input
FEF3H	593CH	Lijn trekken
FEF8H	4039H	Programma eind
FEFDH	40DCH	ERROR controle
FF02H	40FDH	ERROR controle
FF07H	4128H	Hoofdlus "Ok"
FF0CH	4134H	Hoofdlus
FF11H	41A8H	Hoofdlus (directe invoer)
FF16H	4237H	Hoofdlus beeindigd
FF1BH	4247H	Hoofdlus beeindigd
FF20H	42B9H	Token vorming
FF25H	4353H	Token vorming
FF2AH	437CH	Token vorming
FF2FH	43A4H	Token vorming
FF34H	44EBH	Token vorming
FF39H	45D1H	"FOR"
FF3EH	4601H	Runlus nieuw statement
FF43H	4646H	Runlus uitvoeren
FF48H	4666H	CHRGTR routine
FF4DH	4821H	"RETURN"
FF52H	4A5EH	"PRINT"
FF57H	4A94H	"PRINT"
FF5CH	4AFFH	"PRINT"
FF61H	4B4DH	"READ/INPUT" fout
FF66H	4C6DH	Expressie evaluatie
FF6BH	4CA6H	Expressie evaluatie
FF70H	4DD9H	Factor evaluatie
FF75H	4F2CH	Factor evaluatie
FF7AH	4F3EH	Factor evaluatie
FF7FH	51C3H	Runlus uitvoeren
FF84H	51CCH	"WIDTH"

FF89H	522EH	"LIST"
FF8EH	532DH	Tokens vertalen
FF93H	543FH	Integer omzetting
FF98H	5514H	Regelnummer pointer
FF9DH	67EEH	Vrije string/geheugen ruimte
FFA2H	5EA9H	Variabele detectie
FFA7H	148AH	PHYDIO routine
FFACH	148EH	FORMAT routine
FFB1H	406FH	Fouten routine
FFB6H	085DH	LPTOUT routine
FFBBH	0884H	LPTSTT routine
FFC0H	79CCH	"SCREEN"
FFC5H	73E5H	"PLAY" statement

Appendix B
Input/Output (I/O) tabellen

I/O-ADRESSEN

```
------+---+-------------------+-----------+--------+
      !   !                   !           !I/O     !
I/O   !R/W!OMSCHRIJVING       !OPMERKINGEN!ADDRESS !FF
ADRES !   !                   !           !DEVICE  !
------+---+-------------------+-----------+--------+
A8H   !W  !PORT A DATA WRITE  !8255 A     !CHINESE !EO
      !R  !PORT A DATA READ   !COMPATIBLE !ROM     !
      !   !                   !           +--------+D8
A9H   !W  !PORT B DATA WRITE  !           !FDC     !
      !R  !PORT B DATA READ   !           +--------+DO
      !   !                   !           !        !
AAH   !W  !PORT C DATA WRITE  !           !        !
      !R  !PORT C DATA READ   !           !        !
      !   !                   !           !        !
ABH   !W  !MODE SET           !           +--------+CO
------+---+-------------------+-----------+        !
      !   !                   !           !        !
AOH   !W  !ADDRESS LATCH      !AY-3-8910  !        !
      !   !                   !COMPATIBLE !        !
A1H   !W  !DATA WRITE         !           !        !
      !   !                   !           +--------+BO
A2H   !R  !DATA READ          !           !        !
------+---+-------------------+-----------+ PPI    !
      !   !                   !           +--------+BO
98H   !W  !V-RAM DATA WRITE   !9918A      !        !
      !R  !V-RAM DATA READ    !COMPATIBLE ! PSG    !
      !   !                   !           +--------+AO
99H   !W  !COMMAND/ADDRESS SET!           !        !
      !R  !STATUS READ        !           ! VDP    !
------+---+-------------------+-----------+--------+98
      !   !                   !           ! PRIN-  !
90H   !W  !STROBE OUTPUT (bO) !LATCH OUTPUT TER    !
      !R  !STATUS INPUT (b1)  !BUSY '1'   +--------+90
      !   !                   !           !        !
91H   !W  !PRINT DATA         !LATCH OUTPUT        !
------+---+-------------------+-----------+--------+
      !   !                   !           ! RS-    !
80H   !W  !DATA WRITE         !i-8251A    ! 232C   !
      !R  !DATA READ          !COMPATIBLE +--------+80
      !   !                   !           ! NIET   !
81H   !W  !COMMAND/MODE SET   !           ! GE-    !
      !R  !STATUS READ        !           ! SPEC.  !
------+---+-------------------+-----------+--------+
                                                    00
```

I/O adressen 80H t/m FFH zijn voor het systeem gebruik. De lege delen zijn ook voor het gebruik door het systeem.
Hoewel deze adressen hier zijn gedefinieerd, doet u er goed aan deze adressen niet te gebruiken. Elk gebruik dient te gebeuren via de MSX BIOS calls. Dit, om ervoor te zorgen, dat de software onafhankelijk blijft van de hardware ontwikkelingen. Een hardware producent kan een modificatie doorvoeren, maar daarmee wel zorgen voor software compatibiliteit via een BIOS support.
De enige uitzondering op deze regel, is de VDP. Locatie 6 en 7 van de MSX ROM bevatten read en write adressen naar een VDP register. De software waarbij het nodig is om erg snel met de VDP te kunnen communiceren, mag de VDP direct adresseren d.m.v. deze ROM routines.

OOH t/m 7FH zijn vrije adressen. Als echter twee randapparaten dezelfde adressen gebruiken, dan mogen ze niet tegelijkertijd aangesproken worden. I/O devices die niet zijn gespecificeerd, zouden moeten worden opgenomen in het memory mapped I/O in het geheugen.

FDC kan in de I/O ruimte worden opgeslagen, u doet er echter beter aan dit ook in het geheugen te doen. Dit maakt het mogelijk meerder FDC interfaces te gebruiken.

PPI BIT ASSIGNMENT (8251-A)

PORT	BIT	I/O	SIGNAL NAME	BESCHRIJVING
A	0	O	CSOL	0000 - 3FFF address slot select signal
	1		CSOH	
	2		CS1L	4000 - 7FFF address slot select signal
	3		CS1H	
	4		CS2L	8000 - BFFF address slot select signal
	5		CS2H	
	6		CS3L	C000 - FFFF address slot select signal
	7		CS3H	
B	0	I		Keyboard return signal
C	0	O	KB0	Keyboard scan signal
	1		KB1	
	2		KB2	
	3		KB3	
	4		CASON	Cassette control signal (L-ON)
	5		CASW	Cassette write signal
	6		CAPS	CAPS lamp signal
	7		SOUND	Sound output by software

PSG BIT INDELING (AY-3-8910)

PORT	BIT	I/O	CONNECTOR	PIN NR.	SIGNAAL(JOYSTICK)
A	0		J3-1 PIN	NR.1	VOORWAARTS
			J4-1 PIN	NR.2	VOORWAARTS
	1		J3-2 PIN	NR.1	ACHTERWAARTS
			J4-2 PIN	NR.2	ACHTERWAARTS
	2		J3-3 PIN	NR.1	LINKS
			J4-3 PIN	NR.2	LINKS
	3		J3-4 PIN	NR.1	RECHTS
			J4-4 PIN	NR.2	RECHTS
	4		J3-6 PIN	NR.1	VUURKNOP A1
			J4-6 PIN	NR.2	VUURKNOP A2
	5		J3-7 PIN	NR.1	VUURKNOP B1
			J4-7 PIN	NR.2	VUURKNOP B2
	6		KEY LAYOUT	NP. NR.4	"H"/"L" NIVEAU
	7		TAPE READ		
B	0		J3-6 PIN	NR.3	--
	1		J3-7 PIN	NR.3	! "H" NIVEAU
	2		J4-6 PIN	NR.3	!
	3		J4-7 PIN	NR.3	--
	4		J3-8 PIN		
	5		J4-8 PIN		
	6		POORT A(INPUT SELECT)		
	7		KANA LAMP("L"==>>AAN)		

NR.1 Ingeschakeld, als bit 6 van poort B laag is (0) bij het gebruik van joystick1.
NR.2 Ingeschakeld, als bit 6 van poort B hoog is (1) bij het bebruik van joystick1.
NR.3 Veroorzaken een "H" niveau als gebruikt voor output.
NR.4 JIS layout - "H" niveau, letter layout - "L" niveau.

P.S. PIN5 +5V
PIN9 GND

CARTRIDGE SLOT PINBESCHRIJVING

PIN NR.	NAAM	BESCHRIJVING
1	CS1	ROM 4000H-7FFFH selected signal
2	CS2	ROM 8000H-BFFFH selected signal
3	CS12	ROM 4000H-BFFFH selected signal (voor 256K ROM)
4	SLTSL	Slot select signal
5	RESERVED	For futer use only (don't use it!)
6	RFSH	Refresh signal
7	WAIT	Wait signal to CPU
8	INT	Interrupt signal to CPU
9	M1	Fetch cycle signal of CPU
10	BUSDIR	This signal controlls the direction of external data bus buffer when the cartridge is selected. It is low level when the data is sent by the cartridge.
11	IORQ	I/O request signal
12	MERQ	Memory request signal
13	WR	Write signal
14	RD	Read signal
15	RESET	System reset signal
16	RESERVED	For future use only. (don't use it!)
17-32	AO-A15	Address bus
33-40	DO-D7	Data bus
41	GND	Ground
42	CLOCK	CPU clock 3.579MHz
43	GND	Ground
44,46	SW1,SW2	Insert/Remove detect for protection
45,47	+5V	+5V power supply
48	+12V	+12V power supply
49	SOUNDIN	Sound input (-5dbm)
50	-12V	-12V power supply

Appendix C
De Z80 instruktieset

CODE	INSTRUKTIE	CODE	INSTRUKTIE
8E	ADC A,(HL)	DD 39	ADD IX,SP
DD 8E d	ADC A,(IX+d)	FD 09	ADD IY,BC
FD 8E d	ADC A,(IY+d)	FD 19	ADD IY,DE
8F	ADC A,A	FD 29	ADD IY,IY
88	ADC A,B	FD 39	ADD IY,SP
89	ADC A,C		
8A	ADC A,D	A6	AND(HL)
8B	ADC A,E	DD A6 d	AND(IX+d)
8C	ADC A,H	FD A6 d	AND(IY+d)
8D	ADC A,L	A7	AND A
CE d	ADC A,d	A0	AND B
ED 4A	ADC HL,BC	A1	AND C
ED 5A	ADC HL,DE	A2	AND D
ED 6A	ADC HL,HL	A3	AND E
ED 7A	ADC HL,SP	A4	AND H
		A5	AND L
86	ADD A,(HL)	E6 d	AND d
DD 86 d	ADD A,(IX+d)		
FD 86 d	ADD A,(IY+d)	CB 46	BIT 0,(HL)
87	ADD A,A	DD CB d 46	BIT 0,(IX+d)
80	ADD A,B	FD CB d 46	BIT 0,(IY+d)
81	ADD A,C	CB 47	BIT 0,A
82	ADD A,D	CB 40	BIT 0,B
83	ADD A,E	CB 41	BIT 0,C
84	ADD A,H	CB 42	BIT 0,D
85	ADD A,L	CB 43	BIT 0,E
C6 d	ADD A,d	CB 44	BIT 0,H
09	ADD HL,BC	CB 45	BIT 0,L
19	ADD HL,DE		
29	ADD HL,HL	CB 4E	BIT 1,(HL)
39	ADD HL,SP	DD CB d 4E	BIT 1,(IX+d)
DD 09	ADD IX,BC	FD CB d 4E	BIT 1,(IY+d)
DD 19	ADD IX,DE	CB 4F	BIT 1,A
DD 29	ADD IX,IX	CB 48	BIT 1,B
CB 49	BIT 1,C	CB 61	BIT 4,C
CB 4A	BIT 1,D	CB 62	BIT 4,D
CB 4B	BIT 1,E	CB 63	BIT 4,E
CB 4C	BIT 1,H	CB 64	BIT 4,H
CB 4D	BIT 1,L	CB 65	BIT 4,L
CB 56	BIT 2,(HL)	CB 6E	BIT 5,(HL)
DD CB d 56	BIT 2,(IX+d)	DD CB d 6E	BIT 5,(IX+d)

FD CB d 56	BIT 2,(IY+d)
CB 57	BIT 2,A
CB 50	BIT 2,B
CB 51	BIT 2,C
CB 52	BIT 2,D
CB 53	BIT 2,E
CB 54	BIT 2,H
CB 55	BIT 2,L
CB 5E	BIT 3,(HL)
DD CB d 5E	BIT 3,(IX+d)
FD CB d 5E	BIT 3,(IY+d)
CB 5F	BIT 3,A
CB 58	BIT 3,B
CB 59	BIT 3,C
CB 5A	BIT 3,D
CB 5B	BIT 3,E
CB 5C	BIT 3,H
CB 5D	BIT 3,L
CB 66	BIT 4,(HL)
DD CB d 66	BIT 4,(IX+d)
FD CB d 66	BIT 4,(IY+d)
CB 67	BIT 4,A
CB 60	BIT 4,B
CB 79	BIT 7,C
CB 7A	BIT 7,D
CB 7B	BIT 7,E
CB 7C	BIT 7,H
CB 7D	BIT 7,L
DC bb aa	CALL C,aabb
FC bb aa	CALL M,aabb
D4 bb aa	CALL NC,aabb
CD bb aa	CALL aabb
C4 bb aa	CALL NZ,aabb
F4 bb aa	CALL P,aabb
EC bb aa	CALL PE,aabb
E4 bb aa	CALL PO,aabb
CC bb aa	CALL Z,aabb
3F	CCF
BE	CP(HL)
DD BE d	CP(IX+d)
FD BE d	CP(IY+d)
BF	CP A
B8	CP B
B9	CP C
BA	CP D
BB	CP E

FD CB d 6E	BIT 5,(IY+d)
CB 6F	BIT 5,A
CB 68	BIT 5,B
CB 69	BIT 5,C
CB 6A	BIT 5,D
CB 6B	BIT 5,E
CB 6C	BIT 5,H
CB 6D	BIT 5,L
CB 76	BIT 6,(HL)
DD CB d 76	BIT 6,(IX+d)
FD CB d 76	BIT 6,(IY+d)
CB 77	BIT 6,A
CB 70	BIT 6,B
CB 71	BIT 6,C
CB 72	BIT 6,D
CB 73	BIT 6,E
CB 74	BIT 6,H
CB 75	BIT 6,L
CB 7E	BIT 7,(HL)
DD CB d 7E	BIT 7,(HL)
FD CB d 7E	BIT 7,(IY+d)
CB 7F	BIT 7,A
CB 78	BIT 7,B
2F	CPL
27	DAA
35	DEC(HL)
DD 35 d	DEC(IX+d)
FD 35 d	DEC(IY+d)
3D	DEC A
05	DEC B
0B	DEC BC
0D	DEC C
15	DEC D
1B	DEC DE
1D	DEC E
25	DEC H
2B	DEC HL
DD 2B	DEC IX
FD 2B	DEC IY
2D	DEC L
3B	DEC SP
F3	DI
10 d	DJNZ d
FB	EI

BC	CP H
BD	CP L
FE d	CP d
ED A9	CPD
ED B9	CPDR
ED A1	CPI
ED B1	CPIR
ED 46	IM 0
ED 56	IM 1
ED E5	IM 2
ED 78	IN A,(C)
DB d	IN A,(d)
ED 40	IN B,(C)
ED 48	IN C,(C)
ED 50	IN D,(C)
ED 58	IN E,(C)
ED 70	IN F,(C)
ED 60	IN H,(C)
ED 68	IN L,(C)
34	INC(HL)
DD 34 d	INC(IX+d)
FD 34 d	INC(IY+d)
3C	INC A
04	INC B
03	INC BC
0C	INC C
14	INC D
13	INC DE
1C	INC E
24	INC H
23	INC HL
DD 23	INC IX
FD 23	INC IY
2C	INC L
33	INC SP
ED AA	IND
ED BA	INDR
ED 2A	INI
ED B2	INIR
DD 74 d	LD(IX+d),H
DD 75 d	LD(IX+d),l
DD 36 d d1	LD(IX+d),d1
FD 77 d	LD(IY+d),A

E3	EX(SP),HL
DD E3	EX(SP),IX
FD E3	EX(SP),IY
08	EX AF,AF'
EB	EX DE,HL
D9	EXX
76	HALT
E9	JP(HL)
DD E9	JP(IX)
FD E9	JP(IY)
DA bb aa	JP C,aabb
FA bb aa	JP M,aabb
D2 bb aa	JP NC,aabb
C3 bb aa	JP aabb
C2 bb aa	JP NZ,aabb
F2 bb aa	JP P,aabb
EA bb aa	JP PE,aabb
E2 bb aa	JP PO,aabb
CA bb aa	JP Z,aabb
38 d	JR C,d
18 d	JR d
30 d	JR NC,d
20 d	JR NZ,d
28 d	JR Z,d
02	LD(BC),A
12	LD(DE),A
77	LD(HL),A
70	LD(HL),B
71	LD(HL),C
72	LD(HL),D
73	LD(HL),E
74	LD(HL),H
75	LD(HL),L
36 d	LD(HL),d
DD 77 d	LD(IX+d),A
DD 70 d	LD(IX+d),B
DD 71 d	LD(IX+d),C
DD 72 d	LD(IX+d),D
DD 73 d	LD(IX+d),E
7D	LD A,L
3E d	LD A,d
ED 5F	LD A,R
46	LD B,(HL)

```
FD 70 d        LD(IY+d),B
FD 71 d        LD(IY+d),C
FD 72 d        LD(IY+d),D
FD 73 d        LD(IY+d),E
FD 74 d        LD(IY+d),H
FD 75 d        LD(IY+d),L
FD 36 d d1     LD(IY+d),d1

32 bb aa       LD(aabb),A
ED 43 bb aa    LD(aabb),BC
ED 53 bb aa    LD(aabb),DE
22 bb aa       LD(aabb),HL
DD 22 bb aa    LD(aabb),IX
FD 22 bb aa    LD(aabb),IY
ED 73 bb aa    LD(aabb),SP

0A             LD A,(BC)
1A             LD A,(DE)
7E             LD A,(HL)
DD 7E d        LD A,(IX+d)
FD 7E d        LD A,(IY+d)
3A bb aa       LD A,(aabb)
7F             LD A,A
78             LD A,B
79             LD A,C
7A             LD A,D
7B             LD A,E
7C             LD A,H
ED 57          LD A,I
57             LD D,A
50             LD D,B
51             LD D,C
52             LD D,D
53             LD D,E
55             LD D,L
16 d           LD D,d

ED 5B bb aa    LD DE,(aabb)
11 bb aa       LD DE,aabb

4E             LD E,(HL)
DD 5E d        LD E,(IX+d)
FD 5E d        LD E,(IY+d)
5F             LD E,A
58             LD E,B
59             LD E,C
5A             LD E,D
5B             LD E,E
5C             LD E,H
```

```
DD 46 d        LD B,(IX+d)
FD 46 d        LD B,(IY+d)
47             LD B,A
40             LD B,B
41             LD B,C
42             LD B,D
43             LD B,E
44             LD B,H
45             LD B,L
06 d           LD B,d

ED 4B bb aa    LD BC,(aabb)
01 bb aa       LD BC,aabb

4E             LD C,(HL)
DD 4E d        LD C,(IX+d)
FD 4E d        LD C,(IY+d)
4F             LD C,A
48             LD C,B
49             LD C,C
4A             LD C,D
4B             LD C,E
4C             LD C,H
4D             LD C,L
0E d           LD C,d

56             LD D,(HL)
DD 56 d        LD D,(IX+d)
FD 56 d        LD D,(IY+d)
2A bb aa       LD HL,(aabb)
21 bb aa       LD HL,aabb

ED 47          LD I,A

DD 21 bb aa    LD IX,aabb

FD 2A bb aa    LD IY,(aabb)
FD 21 bb aa    LD IY,aabb

6E             LD L,(HL)
DD 6E d        LD L,(IX+d)
FD 6E d        LD L,(IY+d)
6F             LD L,A
68             LD L,B
69             LD L,C
6A             LD L,D
6B             LD L,E
6C             LD L,H
6D             LD L,L
```

5D	LD E,L
1E d	LD E,d
66	LD H,(HL)
DD 66 d	LD H,(IX+d)
FD 66 d	LD H,(IY+d)
67	LD H,A
60	LD H,B
61	LD H,C
62	LD H,D
63	LD H,E
64	LD H,H
65	LD H,L
26 d	LD H,d
ED 44	NEG
00	NOP
B6	OR(HL)
DD B6 d	OR(IX+d)
FD B6 d	OR(IY+d)
B7	OR A
B0	OR B
B1	OR C
B2	OR D
B3	OR E
B4	OR H
B5	OR L
F6 d	OR d
ED BB	OTDR
ED B3	OTIR
ED 79	OUT(C),A
ED 41	OUT(C),B
ED 49	OUT(C),C
ED 51	OUT(C),D
ED 59	OUT(C),E
ED 61	OUT(C),H
ED 69	OUT(C),L
D3 d	OUT(d),A
ED AB	OUTD
ED A3	OUTI
F1	POP AF
C1	POP BC
D1	POP DE
E1	POP HL

2E d	LD L,d
ED 4F	LD R,A
ED 7B bb aa	LD SP,(aabb)
F9	LD SP,HL
DD F9	LD SP,IX
FD F9	LD SP,IY
31 bb aa	LD SP,aabb
ED A8	LDD
ED B8	LDDR
ED A0	LDI
ED B0	LDIR
DD E1	POP IX
FD E1	POP IY
F5	PUSH AF
C5	PUSH BC
D5	PUSH DE
E5	PUSH HL
DD E5	PUSH IX
FD E5	PUSH IY
CB 86	RES 0,(HL)
DD CB d 86	RES 0,(IX+d)
FD CB d 86	RES 0,(IY+d)
CB 87	RES 0,A
CB 80	RES 0,B
CB 81	RES 0,C
CB 82	RES 0,D
CB 83	RES 0,E
CB 84	RES 0,H
CB 85	RES 0,L
CB 8E	RES 1,(HL)
DD CB d 8E	RES 1,(IX+d)
FD CB d 8E	RES 1,(IY+d)
CB 8F	RES 1,A
CB 88	RES 1,B
CB 89	RES 1,C
CB 8A	RES 1,D
CB 8B	RES 1,E
CB 8C	RES 1,H
CB 8D	RES 1,L
CB 96	RES 2,(HL)
DD CB d 96	RES 2,(IX+d)
FD CD d 96	RES 2,(IY+d)

CB 97	RES 2,A
CB 90	RES 2,B
CB 91	RES 2,C
CB 92	RES 2,D
CB 93	RES 2,E
CB 94	RES 2,H
CB 95	RES 2,L
CB 9E	RES 3,(HL)
DD CB d 9E	RES 3,(IX+d)
FD CB d 9E	RES 3,(IY+d)
CB 9F	RES 3,A
CB 98	RES 3,B
CB 99	RES 3,C
CB 9A	RES 3,D
CB B9	RES 3,E
CB 9C	RES 3,H
CB 9D	RES 3,L
CB A6	RES 4,(HL)
DD CB d A6	RES 4,(IX+d)
FD CB d A6	RES 4,(IY+d)
CB A7	RES 4,A
CB A0	RES 4,B
CB A1	RES 4,C
CB A2	RES 4,D
CB A3	RES 4,E
CB A4	RES 4,H
CB A5	RES 4,L
CB AE	RES 5,(HL)
DD CB d AE	RES 5,(IX+d)
FD CB d AE	RES 5,(IY+d)
CB AF	RES 5,A
CB A8	RES 5,B
E0	RET PO
C8	RET Z
ED 4D	RETI
ED 45	RETN
CB 16	RL(HL)
DD CB d 16	RL(IX+d)
FD CB d 16	RL(IY+d)
CB 17	RL A
CB 10	RL B
CB 11	RL C
CB 12	RL D
CB 13	RL E

CB A9	RES 5,C
CB AA	RES 5,D
CB AB	RES 5,E
CB AC	RES 5,H
CB AD	RES 5,L
CB B6	RES 6,(HL)
DD CB d B6	RES 6,(IX+d)
FD CB d B6	RES 6,(IY+d)
CB B7	RES 6,A
CB B0	RES 6,B
CB B1	RES 6,C
CB B2	RES 6,D
CB B3	RES 6,E
CB B4	RES 6,H
CB B5	RES 6,L
CB BE	RES 7,(HL)
DD CB d BE	RES 7,(IX+d)
FD CB d BE	RES 7,(IY+d)
CB BF	RES 7,A
CB B8	RES 7,B
CB B9	RES 7,C
CB BA	RES 7,D
CB BB	RES 7,E
CB BC	RES 7,H
CB BD	RES 7,L
C9	RET
D8	RET C
F8	RET M
D0	RET NC
C0	RET NZ
F0	RET P
E8	RET PE
DD CB d 1E	RR(IX+d)
FD CB d 1E	RR(IY+d)
CB 1F	RR A
CB 18	RR B
CB 19	RR C
CB 1A	RR D
CB 1B	RR E
CB 1C	RR H
CB 1D	RR L
1F	RRA
CB 0E	RRC(HL)
DD CB d 0E	RRC(IX+d)

CB 14	RL H
CB 15	RL L
17	RLA
CB 06	RLC(HL)
DD CB d 06	RLC(IX+d)
FD CB d 06	RLC(IY+d)
CB 07	RLC A
CB 00	RLC B
CB 01	RLC C
CB 02	RLC D
CB 03	RLC E
CB 04	RLC H
CB 05	RLC L
07	RLCA
ED 6F	RLD
CB 1E	RR(HL)
9E	SBC A,(HL)
DD 9E d	SBC A,(IX+d)
FD 9E d	SBC A,(IY+d)
9F	SBC A,A
98	SBC A,B
99	SBC A,C
9A	SBC A,D
9B	SBC A,E
9C	SBC A,H
9D	SBC A,L
DE d	SBC A,d
ED 42	SBC HL,BC
ED 52	SBC HL,DE
ED 62	SBC HL,HL
ED 72	SBC HL,SP
37	SCF
CB C6	SET 0,(HL)
DD CB d C6	SET 0,(IX+d)
FD CB d C6	SET 0,(IY+d)
CB C7	SET 0,A
CB C0	SET 0,B
CB C1	SET 0,C
CB C2	SET 0,D
CB C3	SET 0,E
FD CB d 0E	RRC(IY+d)
CB 0F	RRC A
CB 08	RRC B
CB 09	RRC C
CB 0A	RRC D
CB 0B	RRC E
CB 0C	RRC H
CB 0D	RRC L
0F	RRCA
ED 67	RRD
C7	RST 0
CF	RST 8HEX
D7	RST 10HEX
DF	RST 18HEX
E7	RST 20HEX
EF	RST 28HEX
F7	RST 30HEX
FF	RST 38HEX
CB C9	SET 1,C
CB CA	SET 1,D
CB CB	SET 1,E
CB CC	SET 1,H
CB CD	SET 1,L
CB D6	SET 2,(HL)
DD CB d D6	SET 2,(IX+d)
FD CB d D6	SET 2,(IY+d)
CB D7	SET 2,A
CB D0	SET 2,B
CB D1	SET 2,C
CB D2	SET 2,D
CB D3	SET 2,E
CB D4	SET 2,H
CB D5	SET 2,L
CB DE	SET 3,(HL)
DD CB n DE	SET 3,(IX+d)
FD CB n DE	SET 3,(IY+d)
CB DF	SET 3,A
CB D8	SET 3,B
CB D9	SET 3,C
CB DA	SET 3,D
CB DB	SET 3,E
CB DC	SET 3,H
CB DD	SET 3,L

CB C4	SET 0,H
CB C5	SET 0,H
CB CE	SET 1,(HL)
DD CB d CE	SET 1,(IX+d)
FD CB d CE	SET 1,(IY+d)
CB CF	SET 1,A
CB C8	SET 1,B
CB E3	SET 4,E
CB E4	SET 4,H
CB E5	SET 4,L
CB EE	SET 5,(HL)
DD CB d EE	SET 5,(IX+d)
FD CB d EE	SET 5,(IY+d)
CB EF	SET 5,A
CB E8	SET 5,B
CB E9	SET 5,C
CB EA	SET 5,D
CB EB	SET 5,E
CB EC	SET 5,H
CB ED	SET 5,L
CB F6	SET 6,(HL)
DD CB d F6	SET 6,(IX+d)
FD CB d F6	SET 6,(IY+d)
CB F7	SET 6,A
CB F0	SET 6,B
CB F1	SET 6,C
CB F2	SET 6,D
CB F3	SET 6,E
CB F4	SET 6,H
CB F5	SET 6,L
CB FE	SET 7,(HL)
DD CB d FE	SET 7,(IX+d)
FD CB d FE	SET 7,(IY+d)
CB FF	SET 7,A
CB F8	SET 7,B
CB F9	SET 7,C
CB FA	SET 7,D
CB FB	SET 7,E
CB FC	SET 7,H
CB FD	SET 7,L
CB 3F	SRL A
CB 38	SRL B
CB 39	SRL C
CB 3A	SRL D

CB E6	SET 4,(HL)
DD CB d E6	SET 4,(IX+d)
FD CB d E6	SET 4,(IY+d)
CB E7	SET 4,A
CB E0	SET 4,B
CB E1	SET 4,C
CB E2	SET 4,D
CB 26	SLA(HL)
DD CB d 26	SLA(IX+d)
FD CB d 26	SLA(IY+d)
CB 27	SLA A
CB 20	SLA B
CB 21	SLA C
CB 22	SLA D
CB 23	SLA E
CB 24	SLA H
CB 25	SLA L
CB 36	SLI(HL)
DD CB d 36	SLI(IX+d)
FD CB d 36	SLI(IY+d)
CB 37	SLI A
CB 30	SLI B
CB 31	SLI C
CB 32	SLI D
CB 33	SLI E
CB 34	SLI H
CB 35	SLI L
CB 2E	SRA(HL)
DD CB d 2E	SRA(IX+d)
FD CB d 2E	SRA(IY+d)
CB 2F	SRA A
CB 28	SRA B
CB 29	SRA C
CB 2A	SRA D
CB 2B	SRA E
CB 2C	SRA H
CB 2D	SRA L
CB 3E	SRL(HL)
DD CB d 3E	SRL(IX+d)
FD CB d 3E	SRL(IY+d)
94	SUB H
95	SUB L
D6 d	SUB d

CB 3B	SRL E	AE	XOR(HL)
CB 3C	SRL H	DD AE d	XOR(IX+d)
CB 3D	SRL L	FD AE d	XOR(IY+d)
		AF	XOR A
96	SUB(HL)	A8	XOR B
DD 96 d	SUB(IX+d)	A9	XOR C
FD 96 d	SUB(IY+d)	AA	XOR D
97	SUB A	AB	XOR E
90	SUB B	AC	XOR H
91	SUB C	AD	XOR L
92	SUB D	EE d	XOR d
93	SUB E		

P.S.: d - staat voor displacement byte
aabb - staat voor een 16 bit adres

Appendix D
Romroutines

De Rom-routines kunt U in sommige gevallen gebruiken in Uw BASIC programma's en in elk geval in Uw machinetaal programma's.

In een BASIC programma doet u dat d.m.v. het DEFUSR & HET USR statement.

Voorbeeld: DEFUSR=&H000:X=USR(0)
Veroorzaakt een z.g. warm start.

In een Machinetaal programma doen we dat d.m.v. een z.g. CALL-opdracht en het laden van de juiste registers.

Hieronder volgen de Rom-routines en hun functies:

ADRES	UITLEG
0000H	RESET:Als U deze routine aanspreekt zal de computer een warme start uitvoeren.
0008H	RST 08H:Zelfde als boven, echter nu wordt gekeken of het in register HL staande byte gelijk is aan het byte dat op deze instruktie volgt.
000CH	LD A,(HL) SLOTSELECTIE:Register A bevat het slotnummer. Als resultaat bevatten register A en register E de naar adres HL

verwijzende slot waarde.

0010H VOLGENDE BYTE
Zoekt volgende karakter of basic token HL verwijst naar het volgende karakter.
A bevat het karakter.
Carry vlag gezet als het een getal is.
Zero vlag gezet als het einde is bereikt.

0014H LD (HL),E SLOTSELECTIE:A bevat het slotnummer. HL het adres en E de waarde.

0018H RST 18H: Print de waarde van register A op de 'listdevice' (normaal het beeldscherm).
Als U het adres F416 met een waarde groter dan 0 laadt, zal de printer worden ingeschakeld.

0020H RST 20H :DE en HL worden met elkaar vergeleken.
DE en HL worden niet veranderd.

0028H RST 28H :Geeft het gebruikte type variabele.

0038H RST 38H :Interrupt routine (50 * per seconde).

003EH Functie toetsen nemen hun oude waarde aan.

0041H DISABLES SCREEN
Zet het scherm uit.
De schermkleur neemt de border-

kleur aan.

0044H	ENABLES SCREEN Zet het scherm aan.
0047H	Schrijf naar een VDP register. Register C - bevat het register-nummer. Register B - bevat de data. IN BASIC: VDP(C)=B
004AH	Lees uit het Video RAM. Het adres uit HL wordt in de Accumulator geladen. IN BASIC: A=VPEEK(HL)
004DH	Schrijf naar het Video RAM. De waarde in de Accumulator wordt in HL van het Video RAM geladen. IN BASIC: VPOKE HL,A
0050H	VDP READ INITIALISATIE HL bevat het start adres.
0053H	VDP WRITE INITIALISATIE HL bevat het begin adres.
0056H	SCHRIJFT NAAR VRAM Start in HL Data in A Lengte in BC
0059H	VRAM naar RAM Verplaatst een blok van VRAM naar RAM. Startadres VRAM in HL. Doeladres in RAM in DE. Lengte in BC.

005CH	RAM naar VRAM Startadres RAM in HL Doeladres VRAM in DE Lengte in BC
005FH	Screen mode selectie De Accumulator bevat de waarde van de screenmode. IN BASIC: SCREEN A (MET A=0, 1, 2 OF
0062H	KLEUR INITIALISATIE Kleur als: Voorgrondkleur (F3E9H) Achtergrondkleur (F3EAH) Borderkleur (F3EBH) IN BASIC: COLOR A,B,C
0066H	NON MASKABLE INTERRUPT
0069H	SPRITE INITIALISATIE
006CH	SCREEN 0 INITIALISATIE IN BASIC: SCREEN 0
006FH	SCREEN 1 INITIALISATIE IN BASIC: SCREEN 1
0072H	SCREEN 2 INITIALISATIE IN BASIC: SCREEN 2
0075H	SCREEN 3 INITIALISATIE IN BASIC: SCREEN 3
0078H	INITIALISATIE VDP SCREEN 0
007BH	INITIALISATIE VDP SCREEN 1
007EH	INITIALISATIE VDP SCREEN 2
0081H	INITIALISATIE VDP SCREEN 3

0084H SPRITE VORM TABEL
HL bevat het adres van de sprite vorm tabel.
In reg. A stopt U het sprite nr.

0087H SPRITE ATTRIBUTE TABEL
HL bevat het adres van de sprite attribute tabel.
In reg. A stopt U het sprite nr.

008AH SPRITE GROOTTE
Register A bevat de spritegrootte.
Carry vlag gezet als 16*16 sprite.

008DH Schrijft karakter naar graphics scherm.
Waarde van karakter in Accumulator.

0090H Initialisatie PSG

0093H Schrijf naar een PSG register
Schrijft de waarde van register E in het PSG-register.
met de waarde 'A'.
IN BASIC: SOUND A,E

0096H Lees uit een PSG register
De Accumulator bevat de waarde van het PSG register.

0099H Start de achtergrondmuziek

009CH KEYPRESSED
Kijkt of er een toets wordt ingedrukt.
Zero vlag gezet als toets in

buffer.

009FH	Wacht op een toetsdruk. Wacht net zolang totdat er een toets wordt ingedrukt. De waarde van het karakter wordt in de Accumulator. gestopt. IN BASIC: INPUT$(1)
00A2H	Print inhoud Accumulator op het scherm.
00A5H	Print inhoud Accumulator op de printer.
00A8H	PRINTER CONTROLE FFH in de Accumulator, en Zero-vlag gereset als printer 'ON-LINE'. Zero-vlag geset als printer niet 'ON-LINE'.
00AEH	Invoer tot RETURN/ENTER U kunt een regel laten invoeren tot max. 255 karakters. In het bereik van F55EH tot F65DH. HL bevat het startadres -1.
00B7H	CTRL/STOP CONTROLE Carry-vlag gezet als ingedrukt. IN BASIC: STOP ON:ON STOP GOSUB
00C0H	BEEP Geeft een biep geluidje. IN BASIC: BEEP

00C3H	CLS Wist het scherm en plaast de cursor in de linker bovenhoek. IN BASIC: CLS
00C6H	LOCATE (CURSOR POSITIE) Register H - Regel Register L - Kolom IN BASIC: LOCATE L,H
00C9H	FUNCTIE TOETSEN AAN OF UIT? Zet functietoetsen aan als ze aan moeten.
00CCH	FUNCTIE TOETSEN UIT Schakelt de functietoets-display uit. IN BASIC: KEY OFF
00CFH	FUNCTIE TOETSEN AAN Schakelt de functietoets-display aan. IN BASIC: KEY ON
00D2H	TEKST MODE Zet de computer in tekst mode.
00D5H	OPVRAGEN VAN DE JOYSTICK STATUS Hierdoor komt in de Accumulator de waarde van de richting te staan waarin de joystick wordt bewogen. IN BASIC: A=STICK(A)
00D8H	VUURKNOP ROUTINE In de accumulator staat een waarde. Accumulator 0 - niets Accumulator 255 - toets inge-

drukt

0132H	HOOFDLETTERS AAN/UIT Accumulator = 0: HOOFDLETTERS UIT. Accumulator <>0: HOOFDLETTERS AAN.
0156H	Wist KEYBORD BUFFER
013EH	LEEST HET VDP STATUS REGISTER De Accumulator bevat de waarde (0-255)

Index